TÉCNICO+
FORMAÇÃO AVANÇADA

Deep Learning

Gonçalo M. Correia

GALP 2023

Program for today

Sequence-to-sequence models

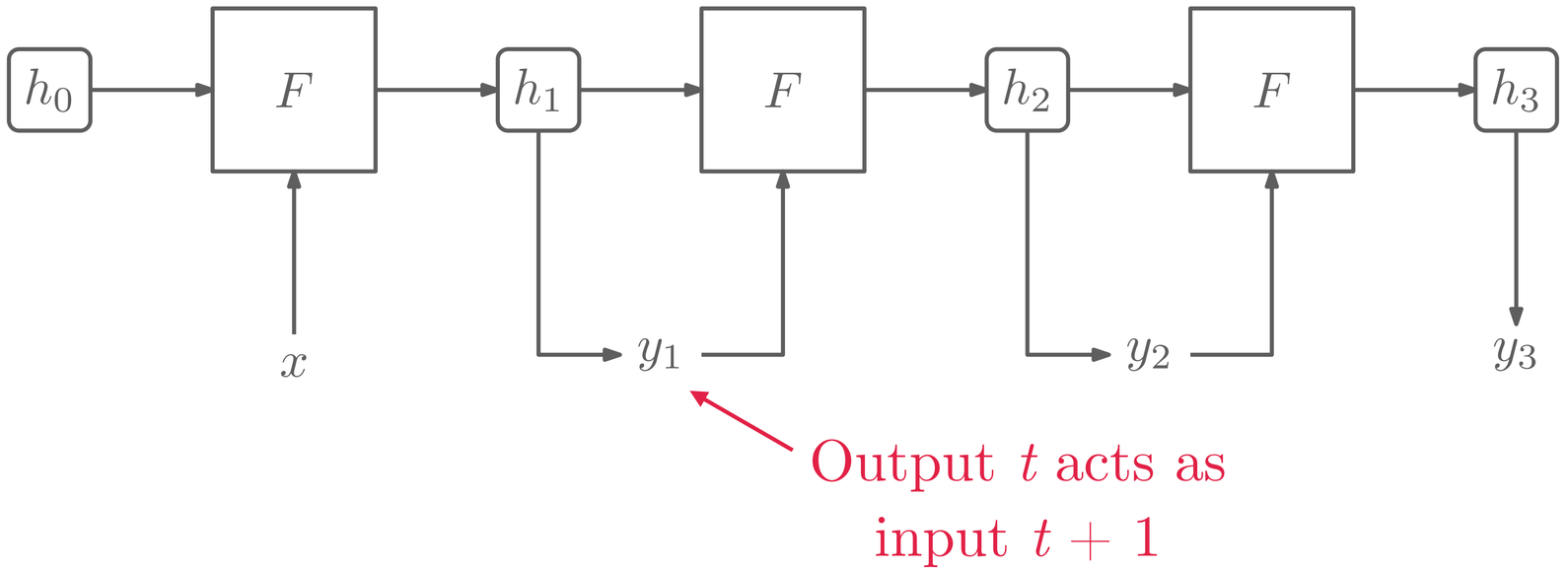
Attention

Transformers

Pre-trained models

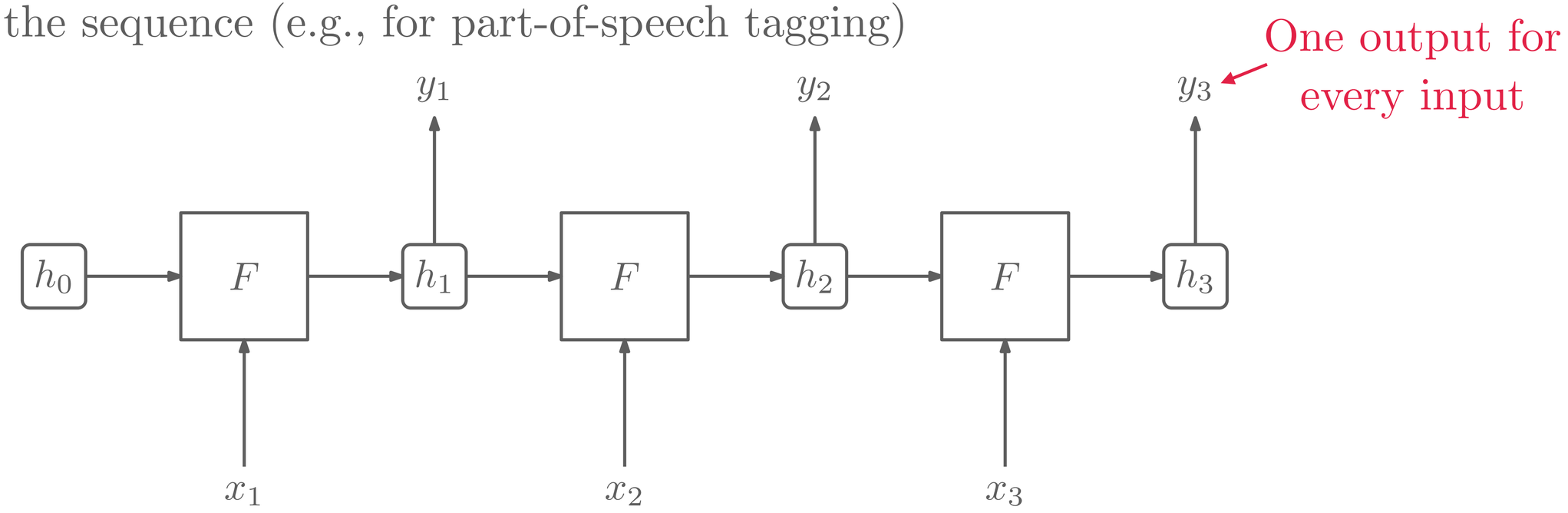
Applications of RNNs

- **Sequence generation:** Network generates a sequence of symbols as an auto-regressive model (e.g., for language generation)



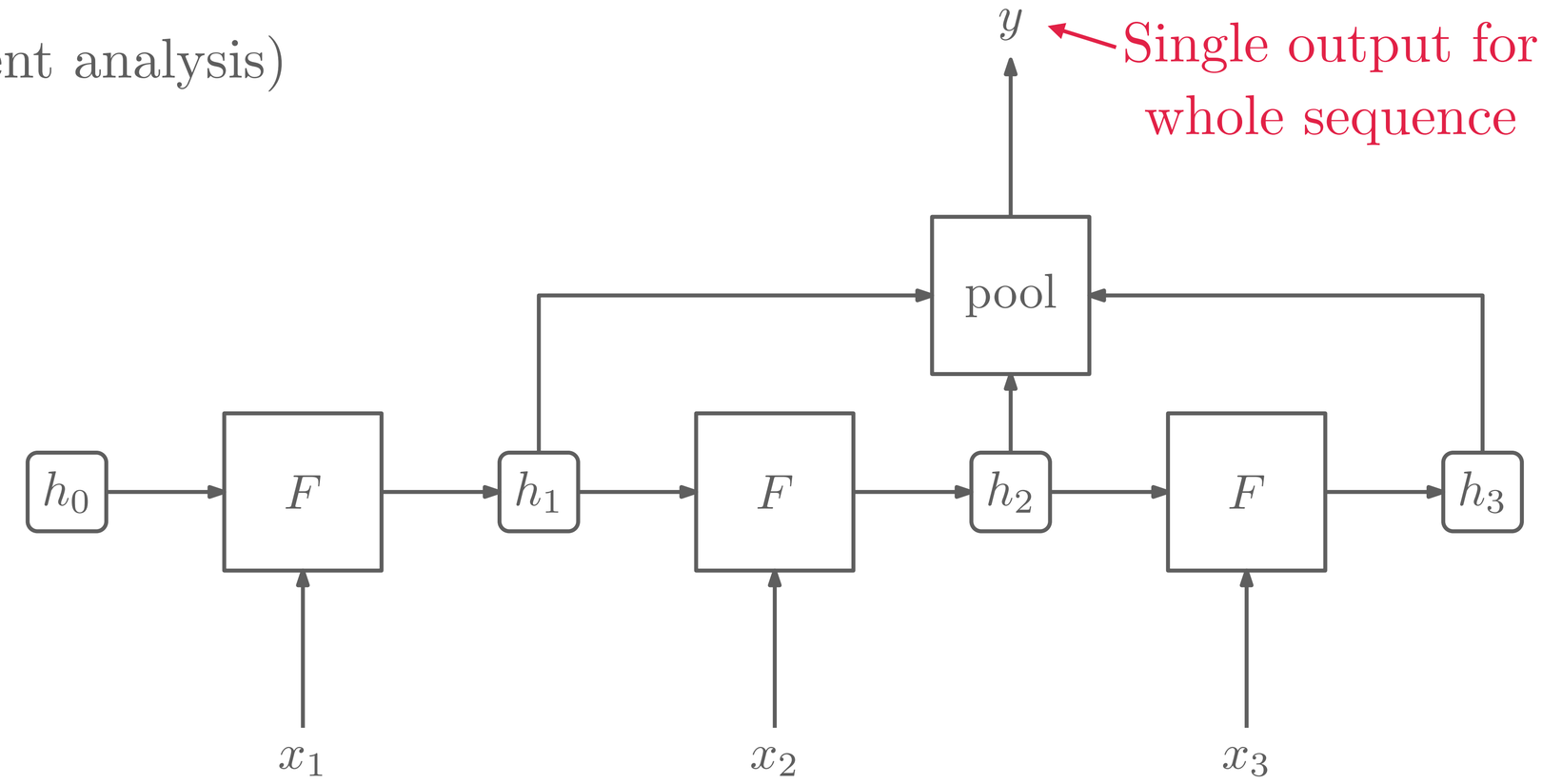
Applications of RNNs

- **Sequence tagging:** Takes a sequence as an input and returns a label/output for each element in the sequence (e.g., for part-of-speech tagging)



Applications of RNNs

- **Pooled classification:** Takes a sequence as an input and returns a single label (e.g., for sentiment analysis)



What if we want to map a sequence to another sequence of possibly different length?

Example

Machine translation

- Suppose we want to translate the Portuguese sentence:

  "*A ilha de Utopia tem duzentas milhas de diâmetro na parte central.*"

  to English:

  "*The island of Utopia is two hundred miles across in the middle part.*"

Source sequence

Target sequence

Statistical machine translation

- For many years, machine translation relied on a statistical approach
- Learn a probabilistic model

$$\mathbb{P}[y \mid x] = \mathbb{P}[x \mid y]\mathbb{P}[y]$$

Translated sentence

Original sentence

Statistical machine translation

- For many years, machine translation relied on a statistical approach
- Learn a probabilistic model

$$\mathbb{P}[y \mid x] = \mathbb{P}[x \mid y]\mathbb{P}[y]$$

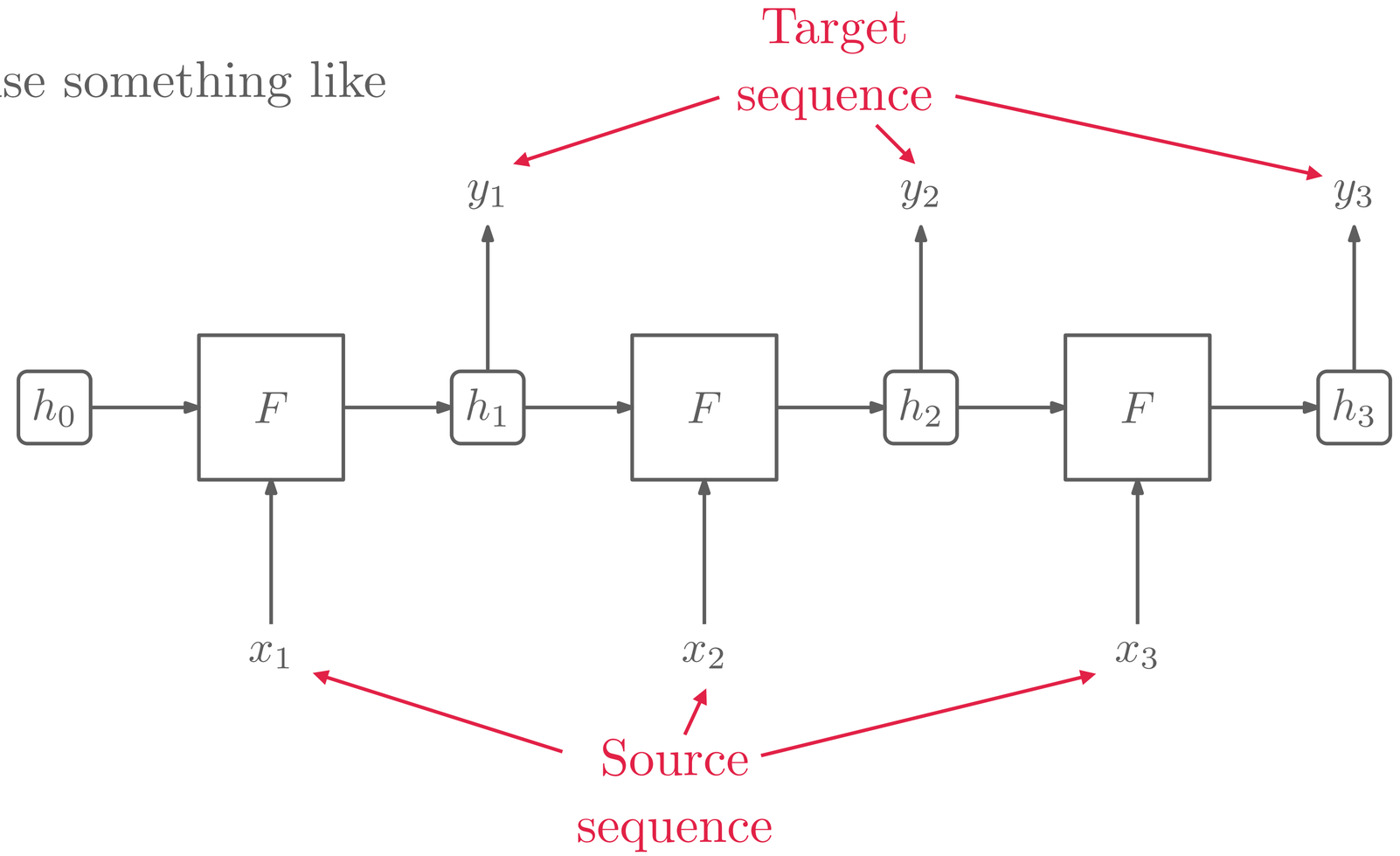
Translation model

Language model

Can we do it using deep learning?

The encoder-decoder architecture

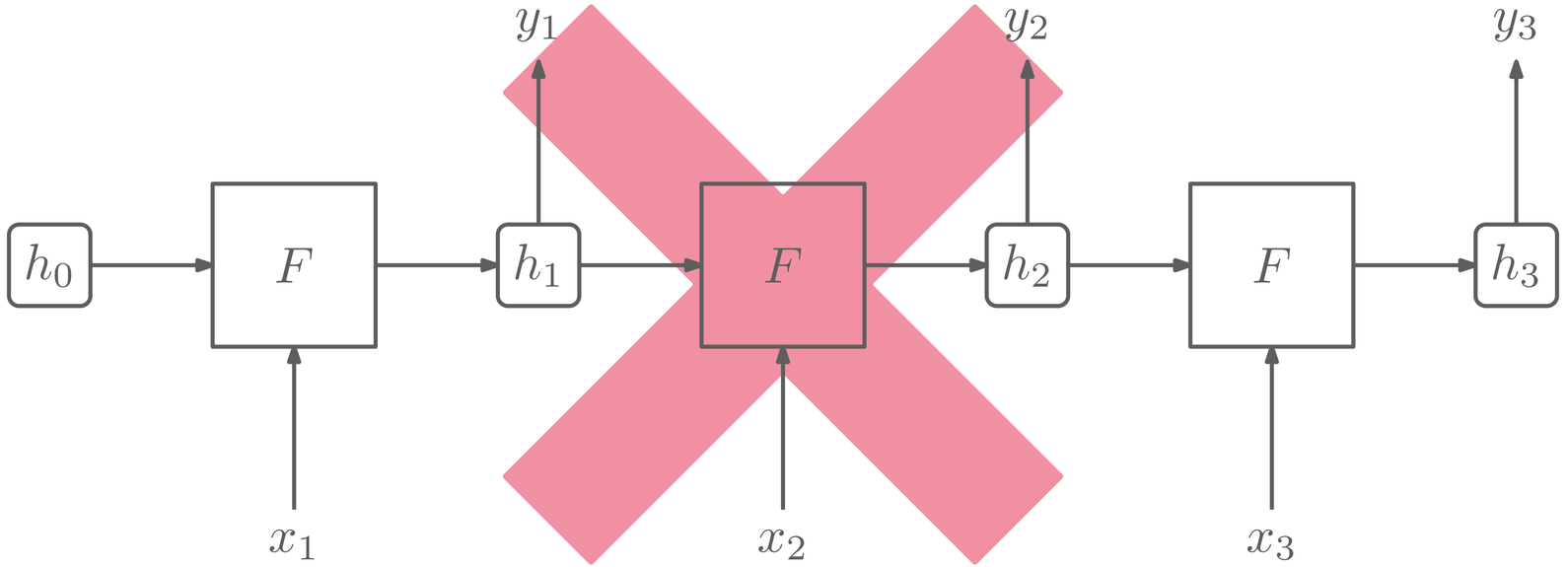
- Could we use something like



The encoder-decoder architecture

- Problems:
  - We often need to process the whole source sequence before producing the output sequence
  - Source and target sequences often have different sizes

The encoder-decoder architecture

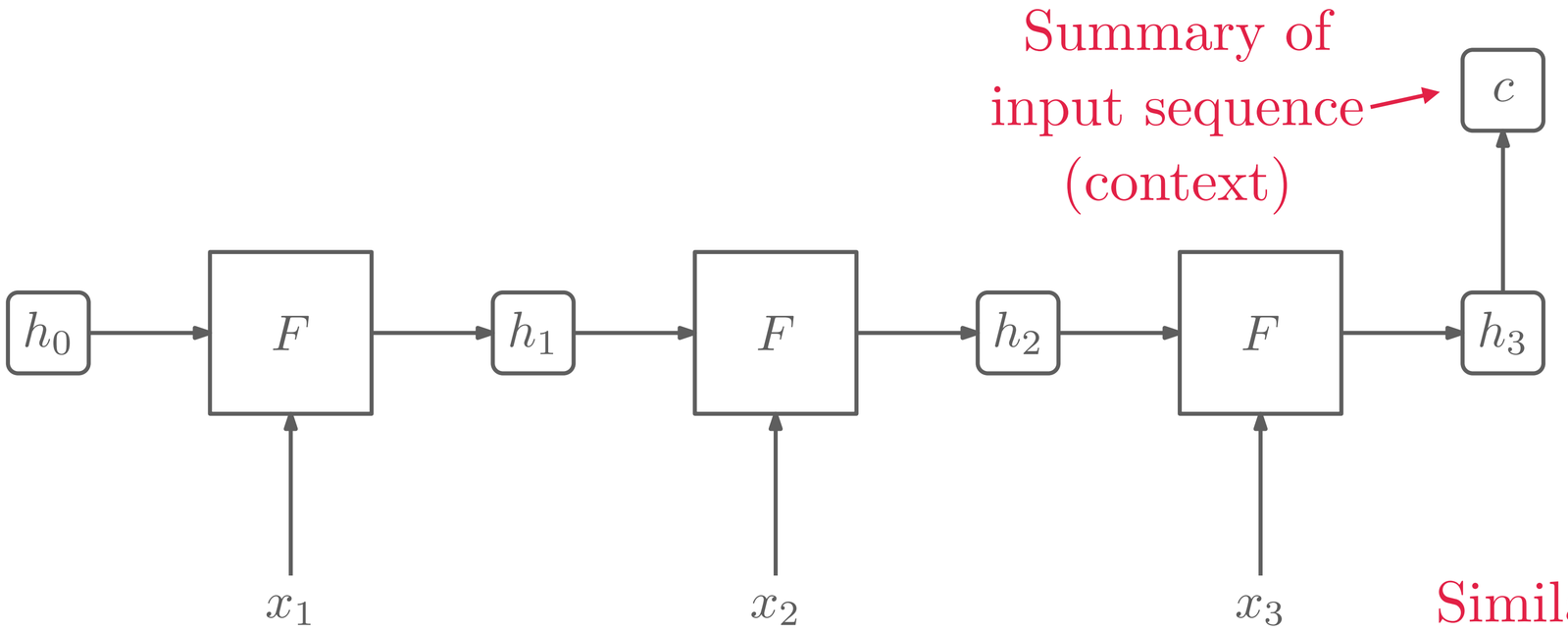


The encoder-decoder architecture

- We divide the process of machine translation in two distinct steps:
  - Process the source sequence
  - Generate the output sequence

The encoder-decoder architecture

- We process the input sequence using the recurrent architecture

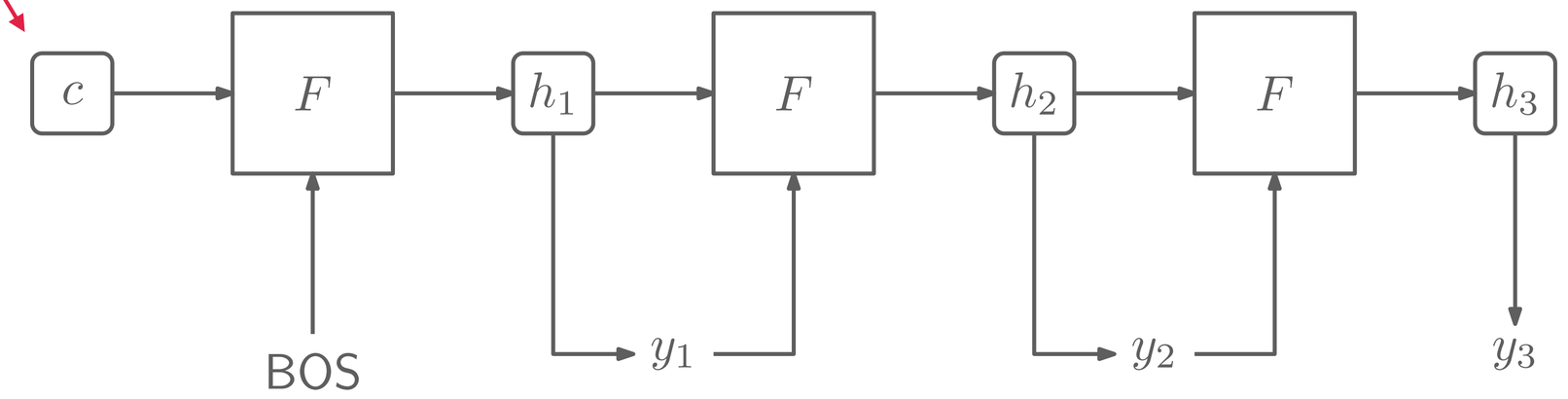


Similar to pooled-classification architecture

The encoder-decoder architecture

- We generate the output sequence using another recurrent architecture

We use the context to bootstrap the generation

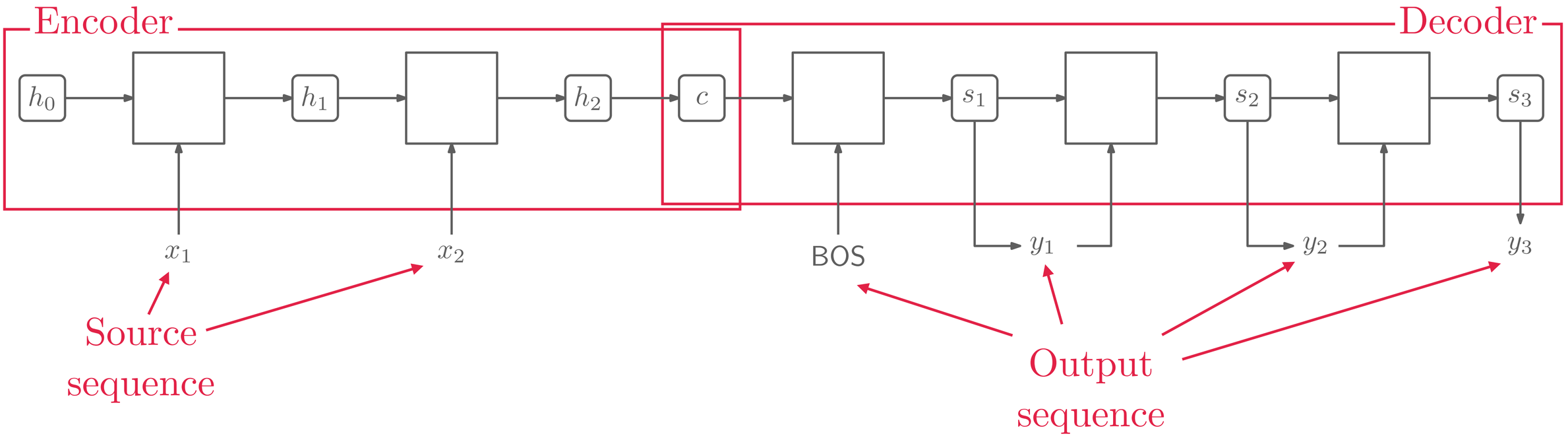


Sequence-generation architecture

The encoder-decoder architecture

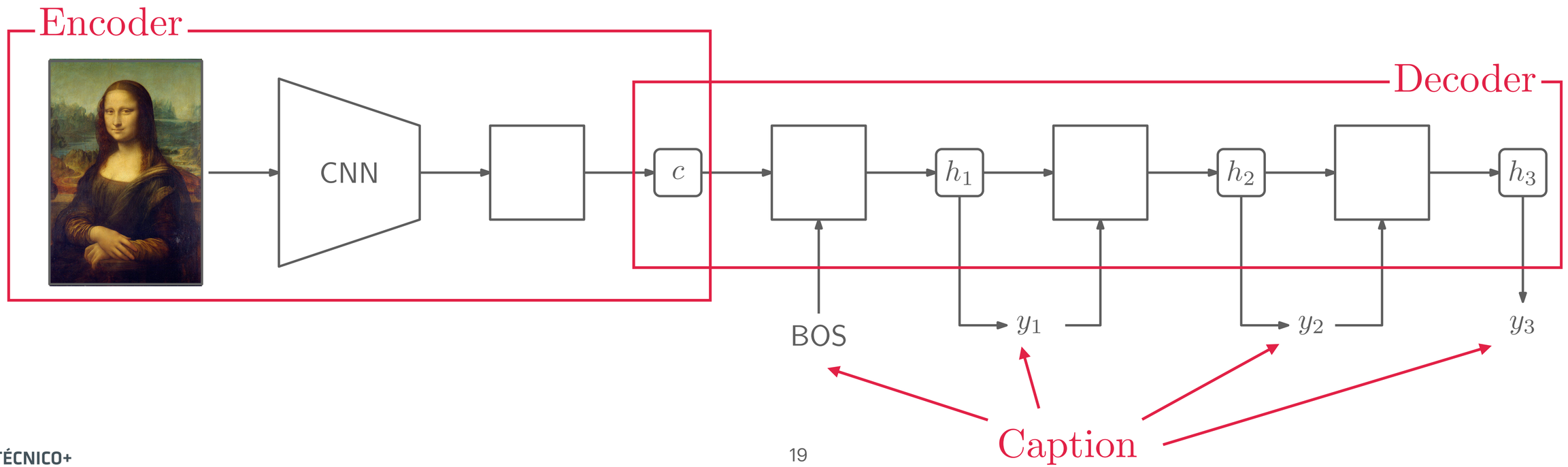
- We divide the process of machine translation in two distinct steps:
  - The encoder processes the source sequence
  - The decoder generates the output sequence conditioned on the input sequence
- The resulting model is also known as a sequence-to-sequence model

The sequence-to-sequence model



The encoder-decoder architecture

- The same basic architecture can be used for image captioning
  - The encoder is a CNN instead of a RNN



Neural machine translation

- Model can be trained end-to-end
- Replaced previous statistical models, developed by teams of engineers during several years
- Leading methods for machine translation since 2016

Challenges

- For long input sequences, the encoder may “forget” the first elements
- Output sequence should be the most likely given the input
  - Greedy generation does not guarantee most likely sequence
  - Generation is usually done using beam search

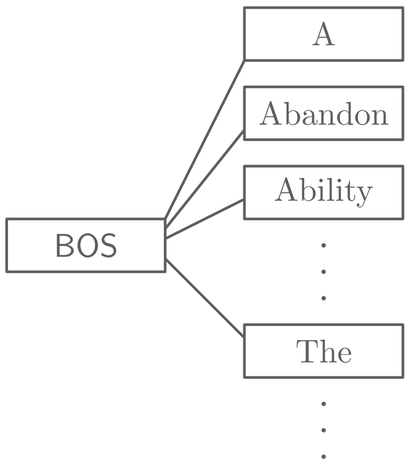
Beam search

- Grows a tree of possible sequences
- At each level, keeps only k best sequences
- E.g., for $k = 2$

BOS

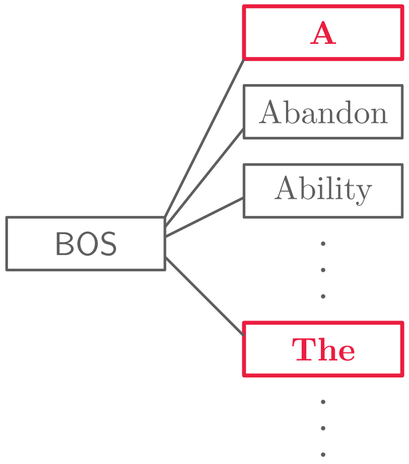
Beam search

- Grows a tree of possible sequences
- At each level, keeps only k best sequences
- E.g., for $k = 2$



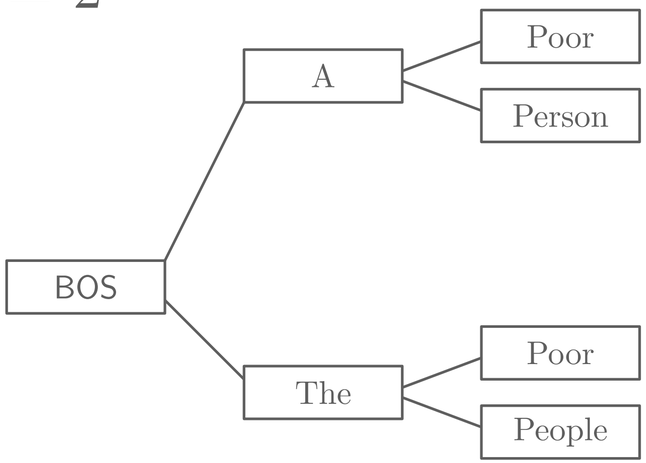
Beam search

- Grows a tree of possible sequences
- At each level, keeps only k best sequences
- E.g., for $k = 2$



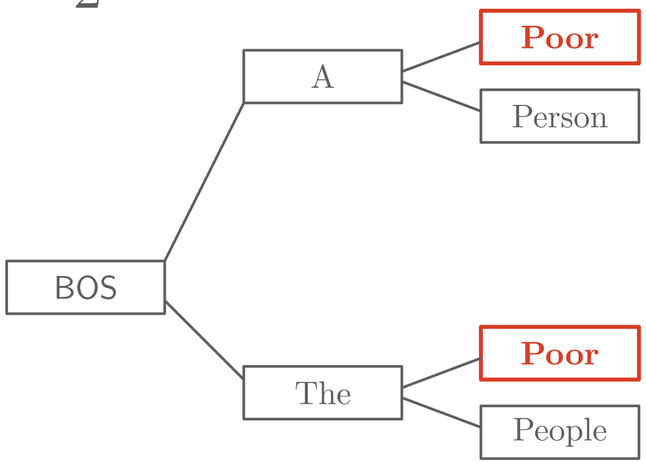
Beam search

- Grows a tree of possible sequences
- At each level, keeps only k best sequences
- E.g., for $k = 2$



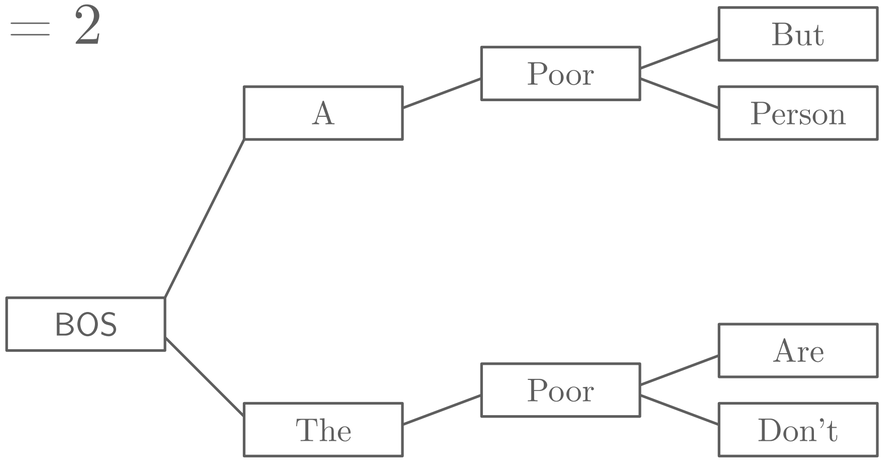
Beam search

- Grows a tree of possible sequences
- At each level, keeps only k best sequences
- E.g., for $k = 2$



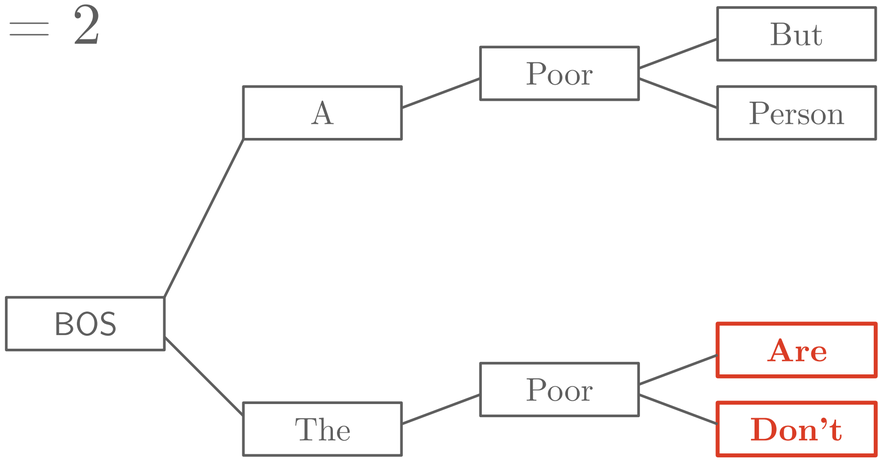
Beam search

- Grows a tree of possible sequences
- At each level, keeps only k best sequences
- E.g., for $k = 2$



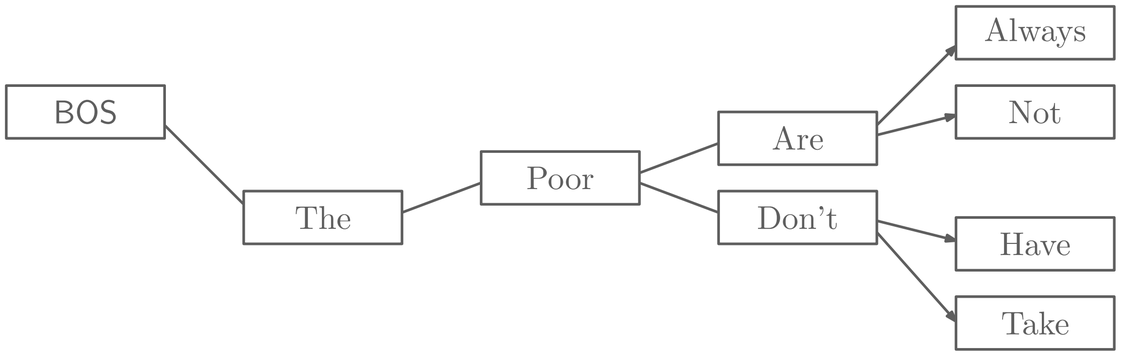
Beam search

- Grows a tree of possible sequences
- At each level, keeps only k best sequences
- E.g., for $k = 2$



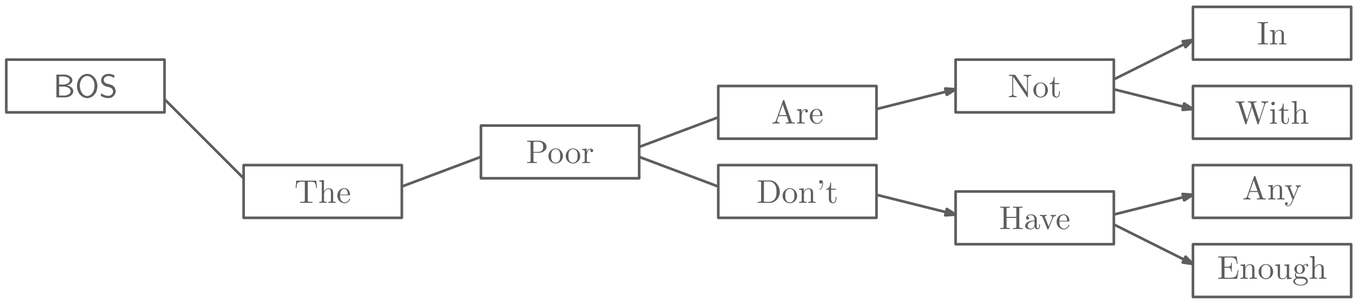
Beam search

- Grows a tree of possible sequences
- At each level, keeps only k best sequences
- E.g., for $k = 2$



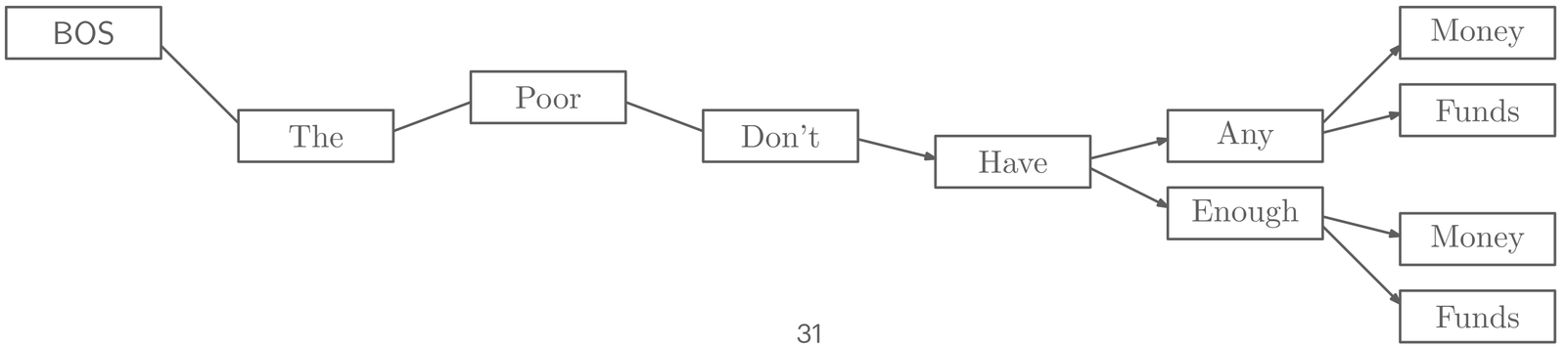
Beam search

- Grows a tree of possible sequences
- At each level, keeps only k best sequences
- E.g., for $k = 2$



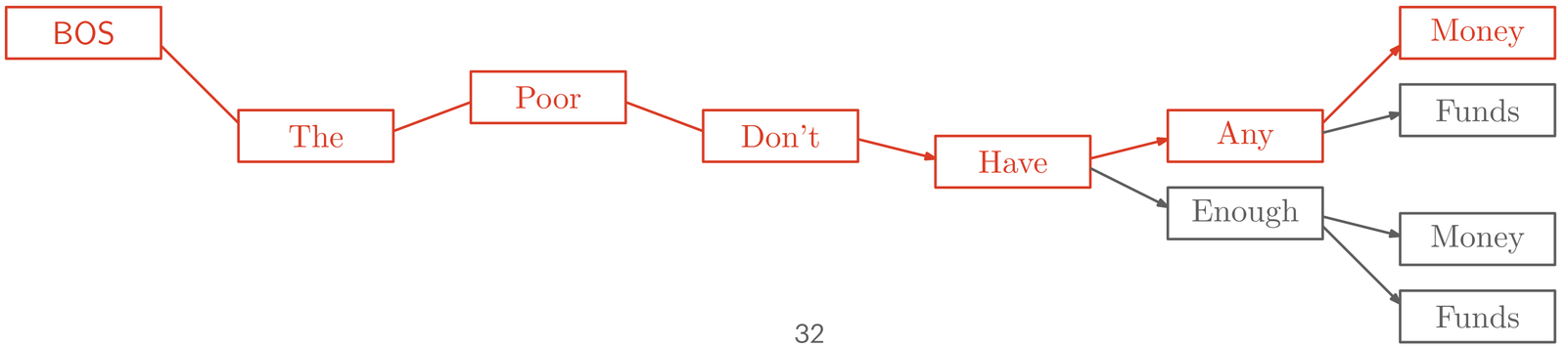
Beam search

- Grows a tree of possible sequences
- At each level, keeps only k best sequences
- E.g., for $k = 2$



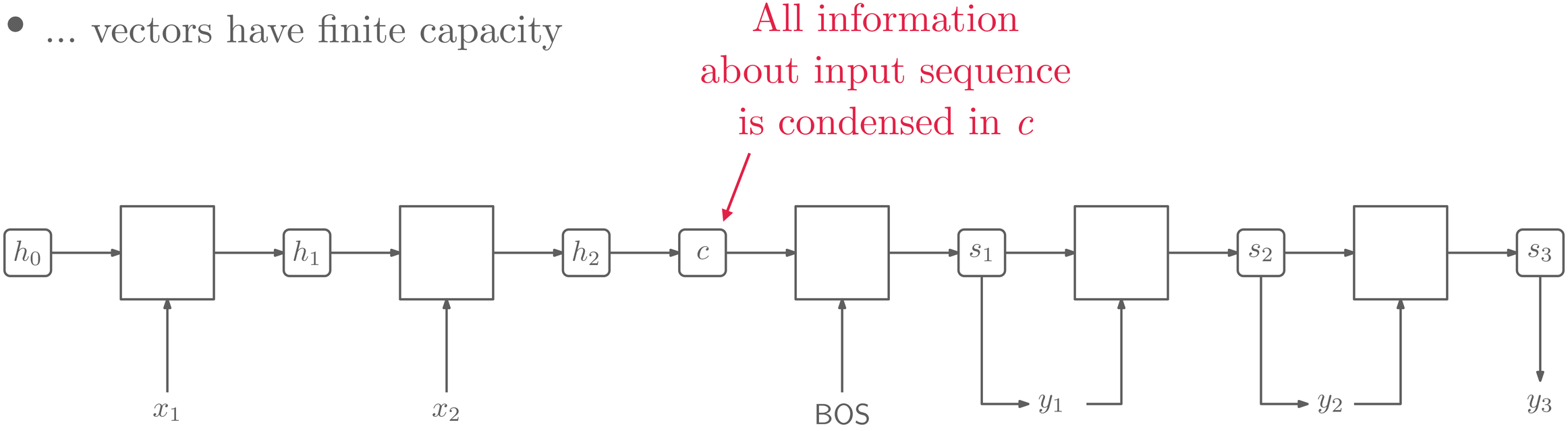
Beam search

- Grows a tree of possible sequences
- At each level, keeps only k best sequences
- E.g., for $k = 2$



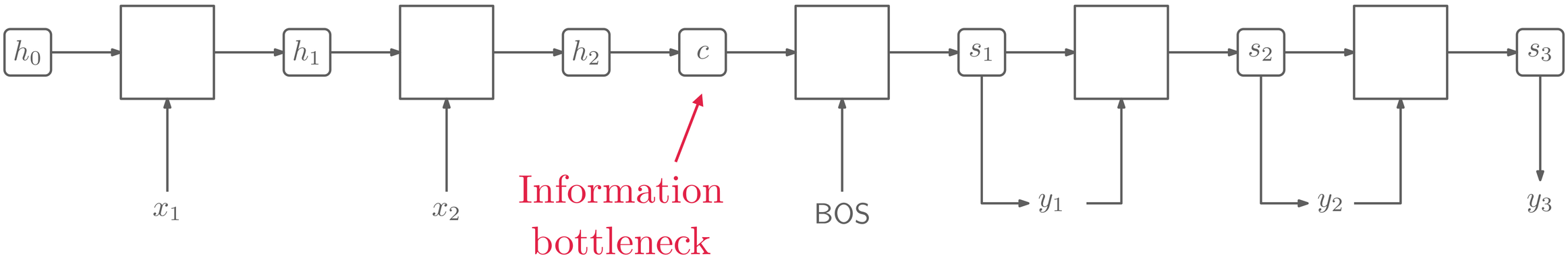
... a more conceptual challenge

- Sentences can have unbounded lengths...
- ... vectors have finite capacity




You can’t cram the meaning of a whole %&$# sentence into a single $&# vector!
Ray Mooney

... a more conceptual challenge

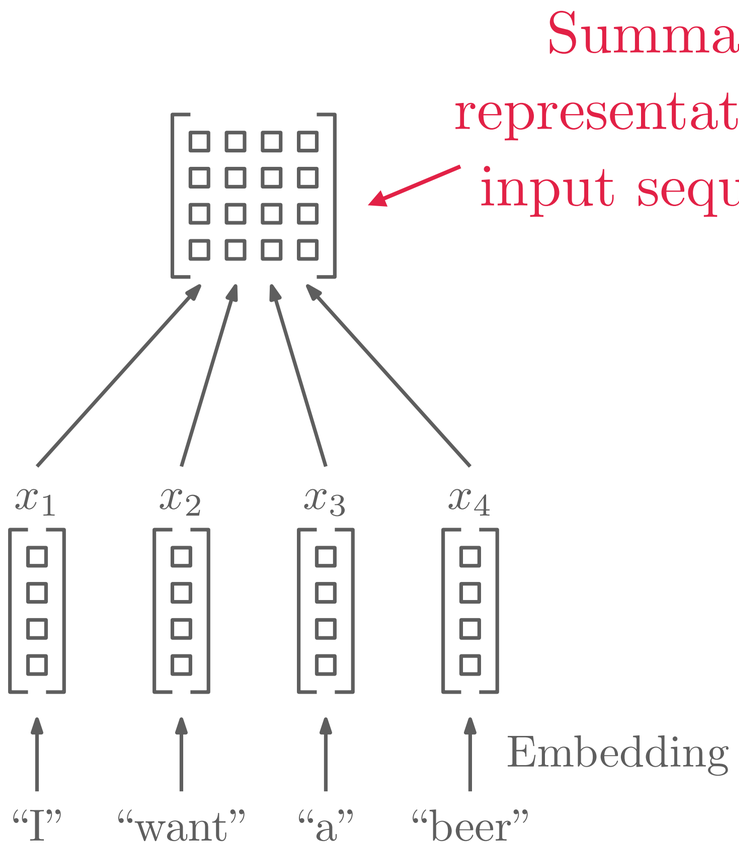


Matrices, not vectors

- What if instead of summarizing the input sequence as a vector, we use a matrix?
  - Fixed number of rows, but number of columns depends on the length of input sequence

What matrix?

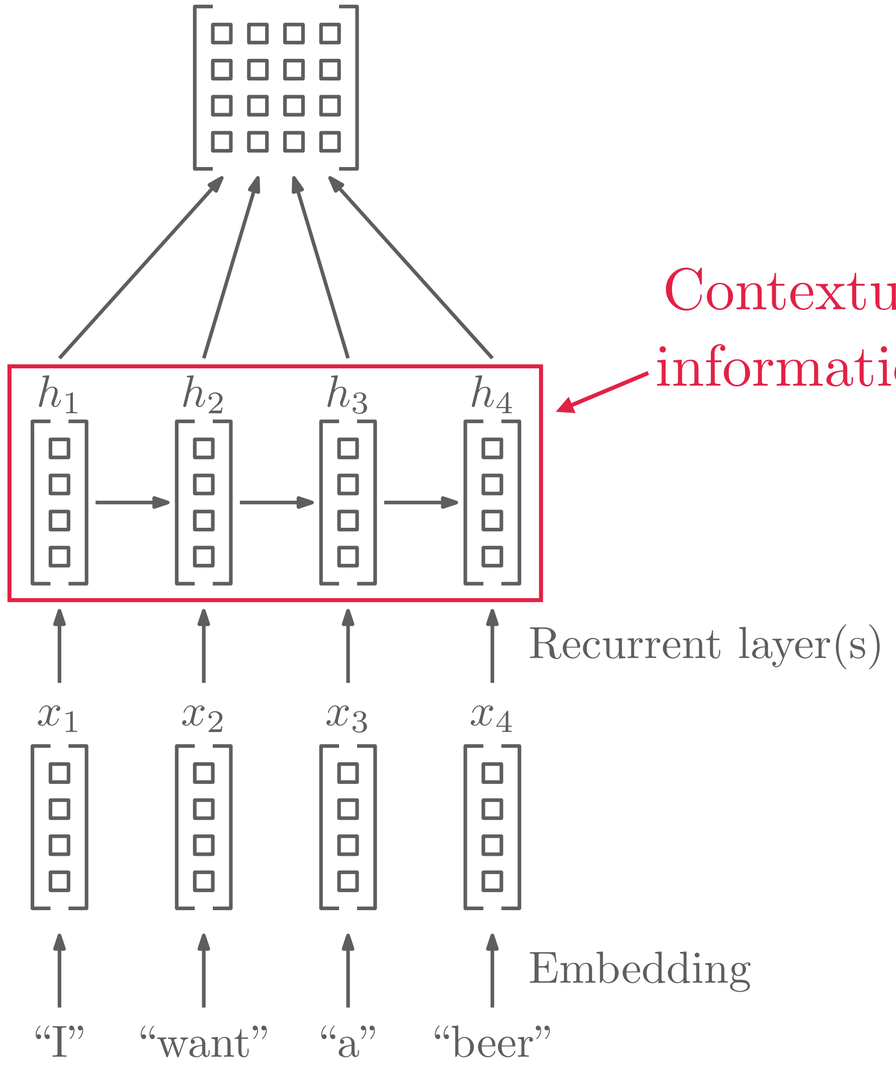
- Alternative n. 1: Embedding matrix
  - We use the embedding vectors as columns



Not a lot of contextual (sequence) information

What matrix?

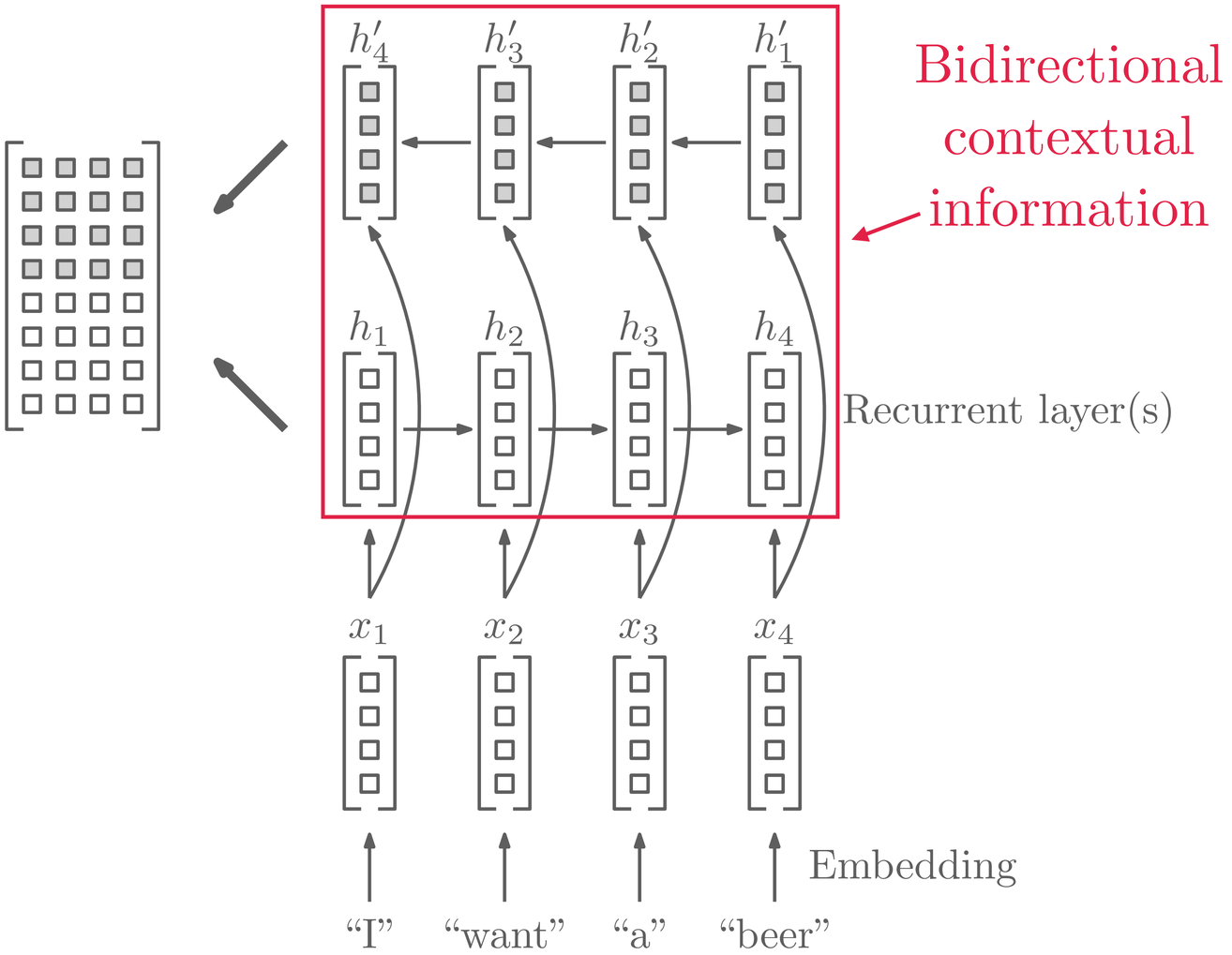
- Alternative n. 2: Hidden vector matrix
  - We use the hidden vectors as columns



Context only looks “back” in time

What matrix?

- Alternative n. 3: Hidden vectors of bi-directional RNN
  - We use the forward and backward hidden vectors as columns



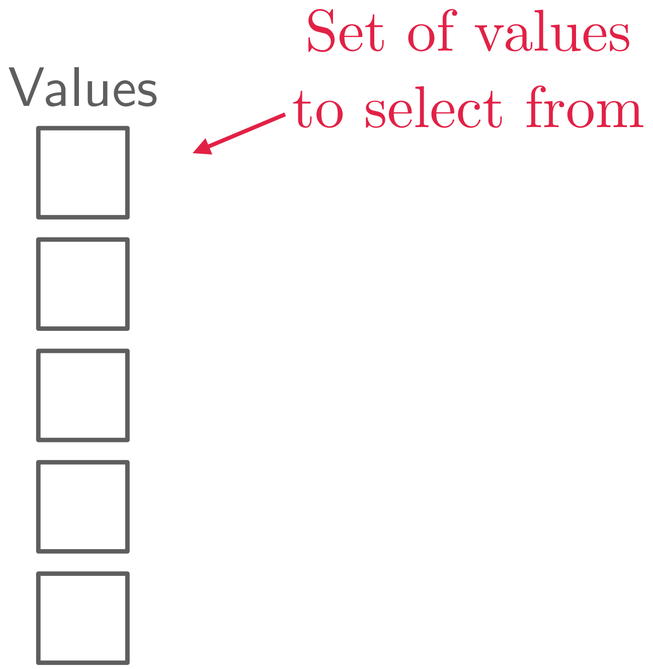
... but...
how do we use the matrix??

Attention

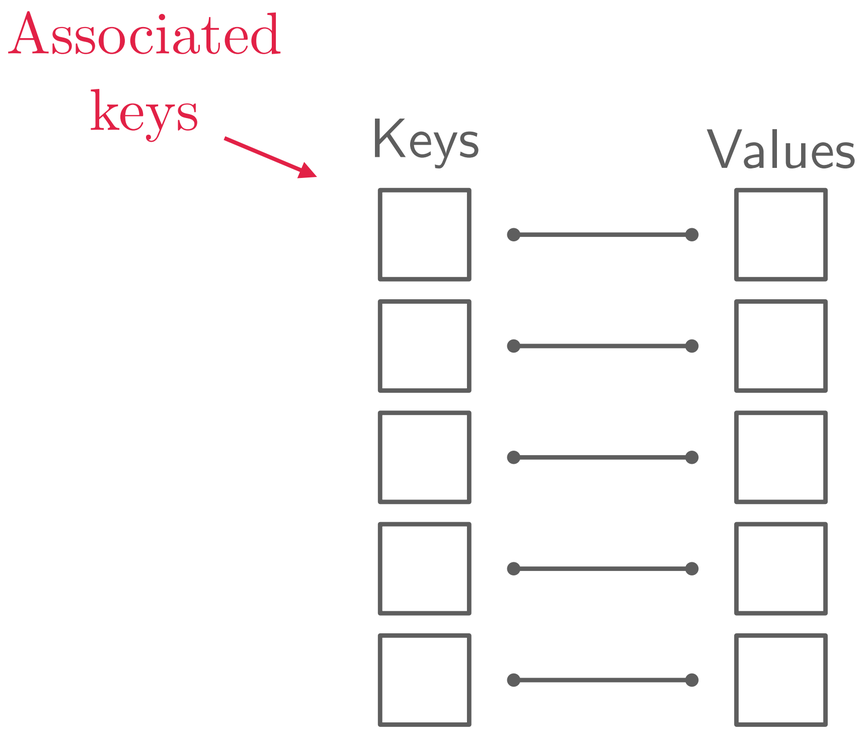
Attention

- Attention is an information selection mechanism
  - We have a set of values
  - Each value has an associated key
  - At selection time, given some query:
    - We compare our query with the different keys
    - Based on similarity between queries and keys, we select a value

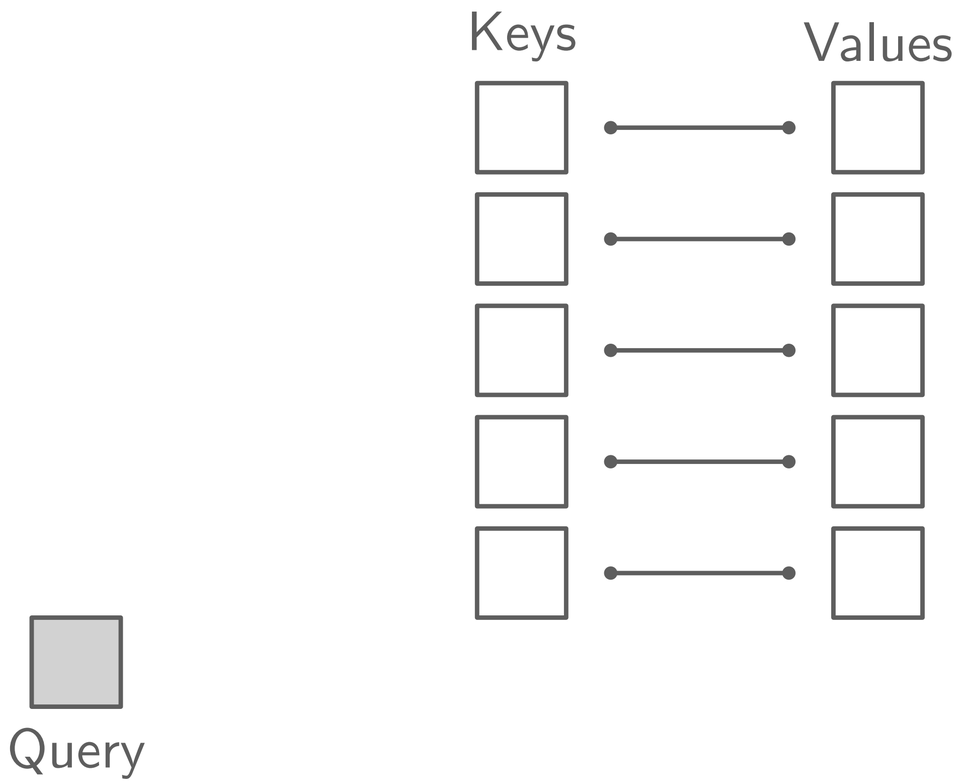
Attention



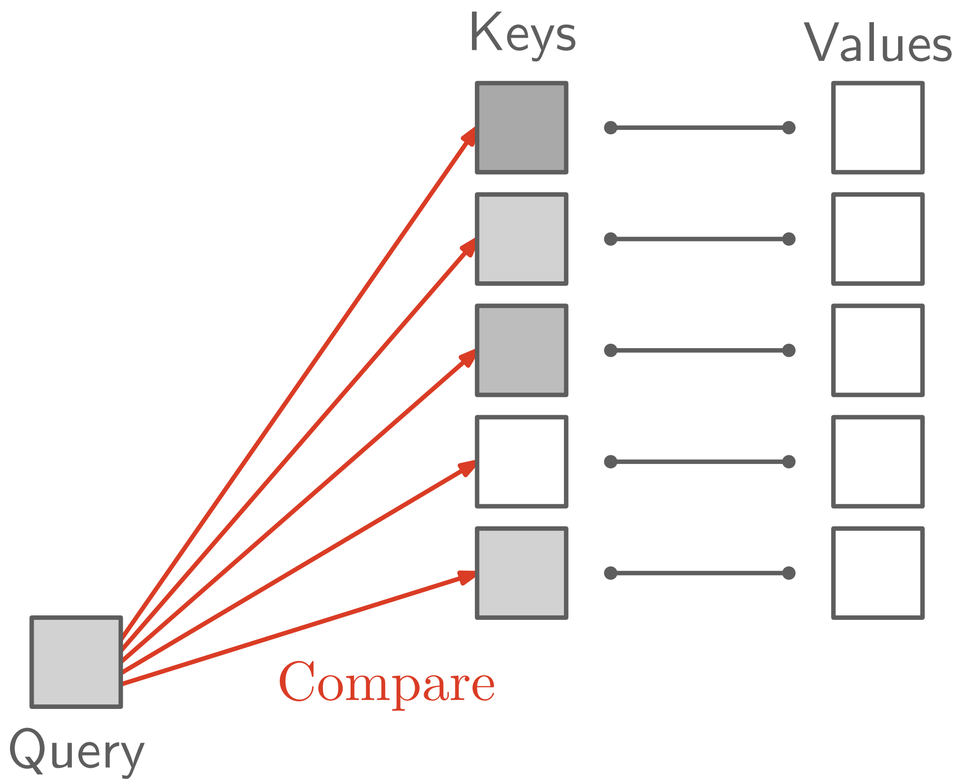
Attention



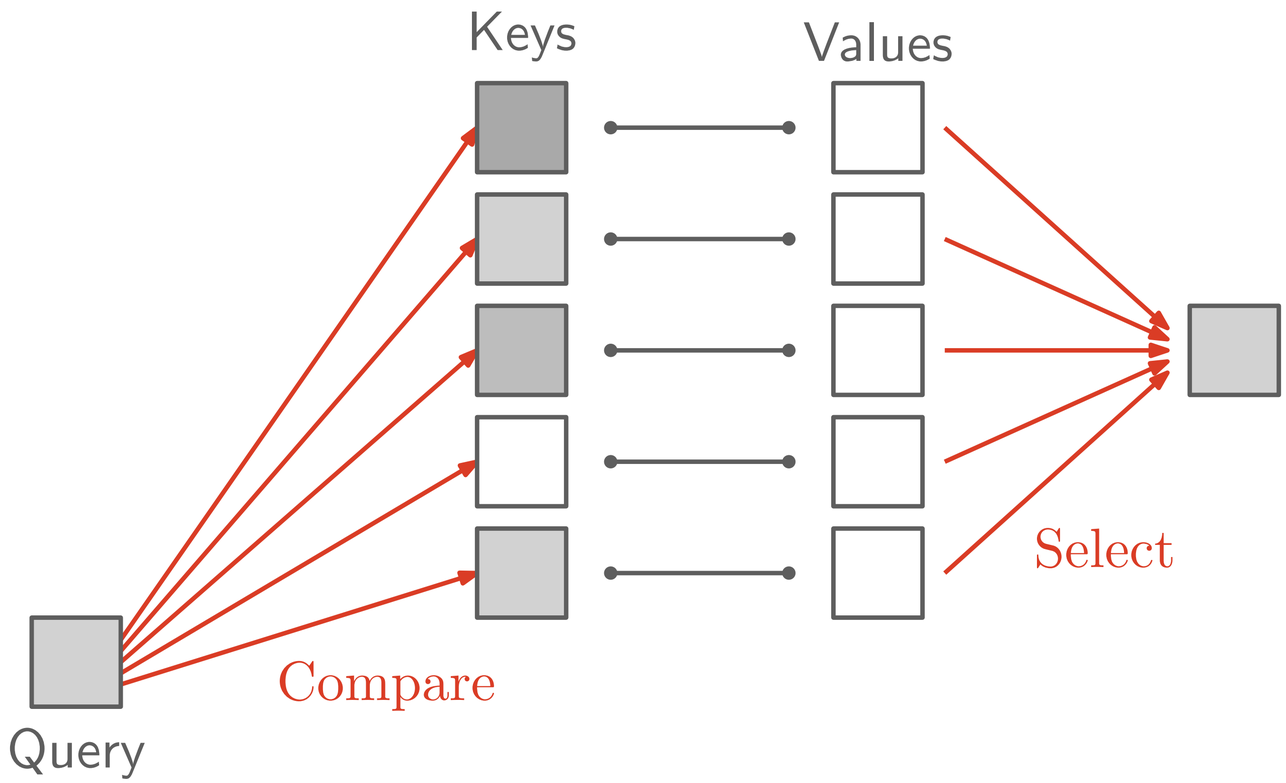
Attention



Attention

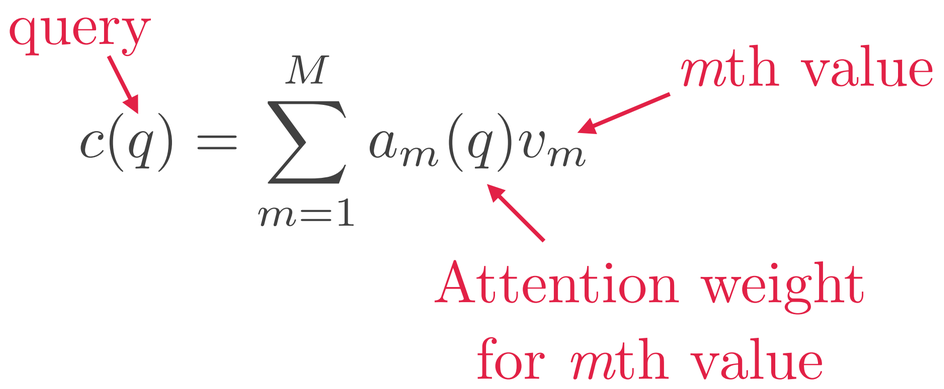


Attention



Attention

- The selection is usually done by weighting the values using attention weights



Attention

- The selection is usually done by weighting the values using attention weights

$$c(q) = \sum_{m=1}^{M} a_m(q) v_m$$

- Attention weights are computed as

Similarity between query and mth key

$$a_m(q) = \frac{\exp(\text{sim}(q, k_m))}{\sum_{m'=1}^{M} \exp(\text{sim}(q, k_{m'}))}$$

Softmax

Attention mechanisms

- Different attention mechanisms arise from different ways of measuring similarity between query and keys

Attention mechanisms

Additive attention:

- Uses a standard fully connected layer to compute similarity

$$\text{sim}(q, k) = w_0^\top F(W_q q + W_k k)$$

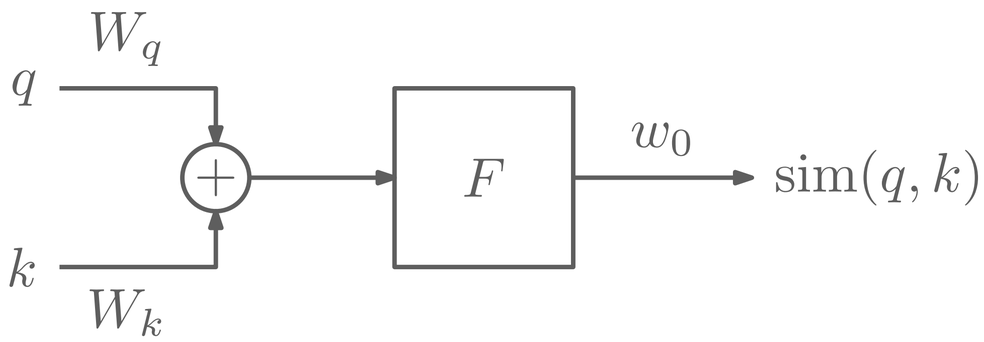
Learned weights

Attention mechanisms

Additive attention:

- Uses a standard fully connected layer to compute similarity

$$\text{sim}(q, k) = w_0^\top F(W_q q + W_k k)$$



Attention mechanisms

Dot-product attention:

- Uses dot-product to compute similarity

$$\text{sim}(q, k) = q^\top k$$

Attention mechanisms

Scaled dot-product attention:

- Uses (scaled) dot-product to compute similarity

$$\text{sim}(q, k) = \frac{q^\top k}{\sqrt{d}}$$

Dimension of k

- Scaling ensures that variance of dot-product does not increase with dimension

In our case...

- Queries: Decoder states, s_t
- Keys: Encoder states, h_t
- Values: Encoder states, h_t

Using attention

- How to incorporate attention in the sequence-to-sequence models?
  - Bahdanau attention
  - Luong attention

Bahnadau attention

- Given the states of the decoder, s_1, s_2, ...
  - Compute attention weights,

$$a_t(s_{t-1}) = \text{softmax}(\text{sim}(s_{t-1}, H))$$

Matrix of encoder states

  - Compute context,

$$c_t = H \cdot a_t$$

  - Update RNN,

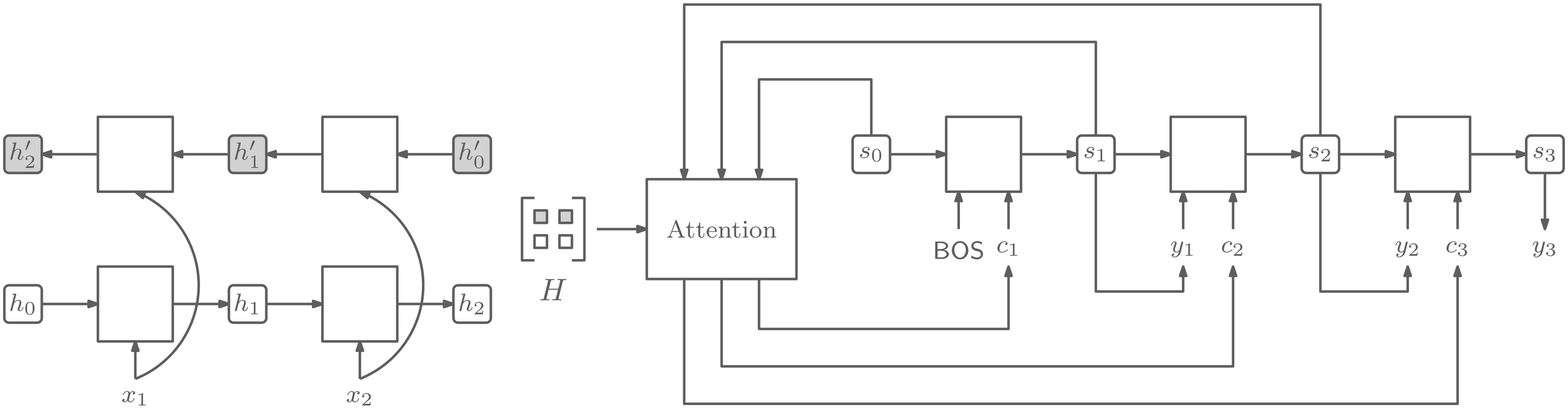
$$s_t = \text{RNN}(s_{t-1}, y_{t-1}, c_t)$$

Context is used as input to the RNN

  - Compute output,

$$y_t \sim \mathbb{P}[y \mid s_t]$$

Bahnadau attention



Luong attention

- Given the states of the decoder, s_1, s_2, ...
  - Compute attention weights,

$$a_t(s_t) = \text{softmax}(\text{sim}(s_t, H))$$

Current state
(not previous state)

  - Compute context,

$$c_t = H \cdot a_t$$

  - Compute pseudo-state,

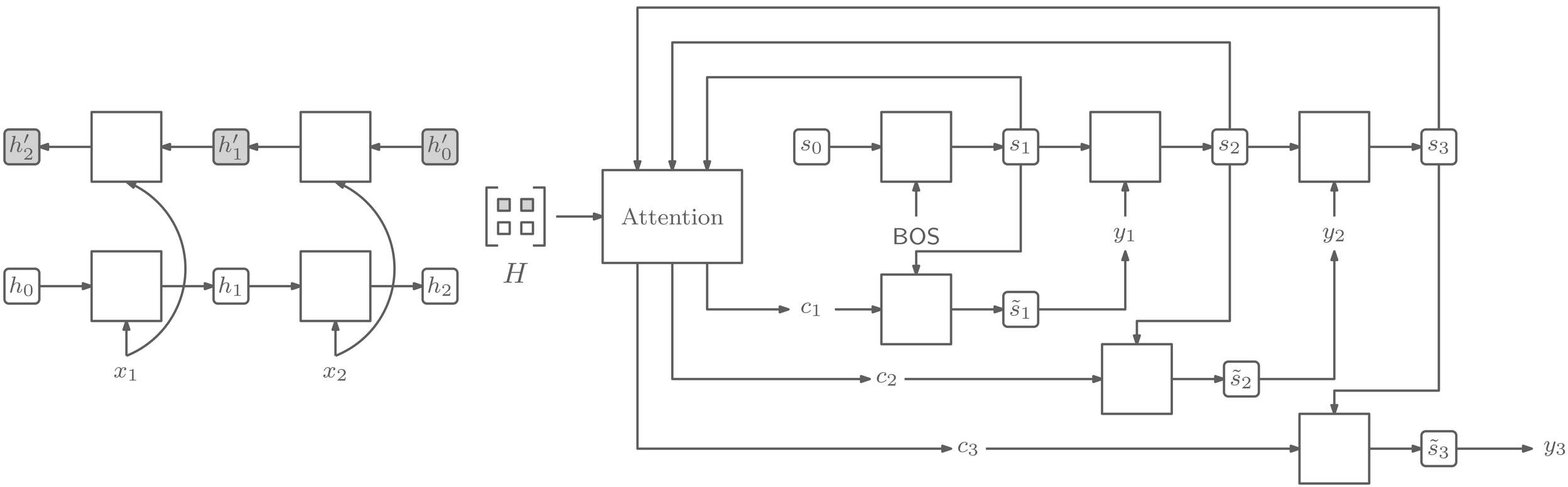
$$\tilde{s}_t = W_y \tanh(W_c[c_t, s_t])$$

Context is used
as input to FC layer

  - Compute output,

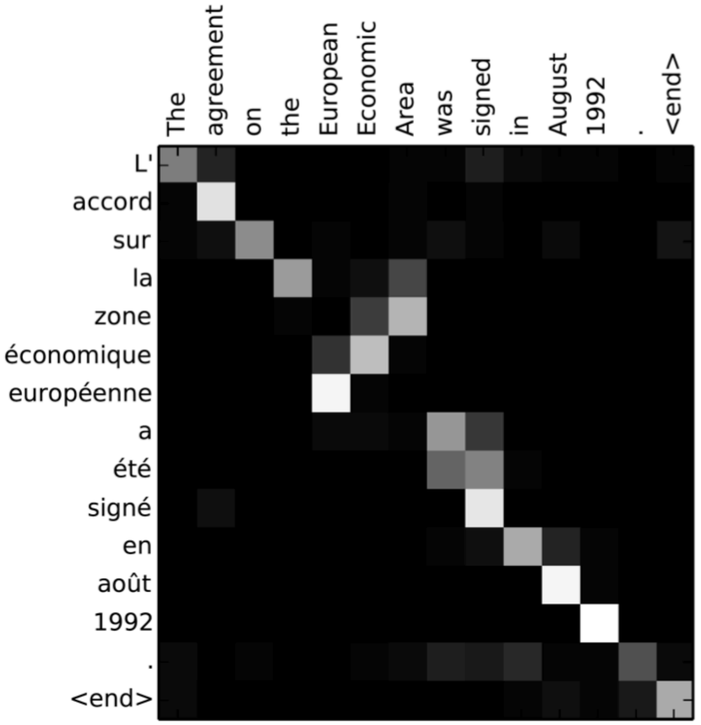
$$y_t \sim \mathbb{P}[y \mid \tilde{s}_t]$$

Luong attention



Attention in NMT

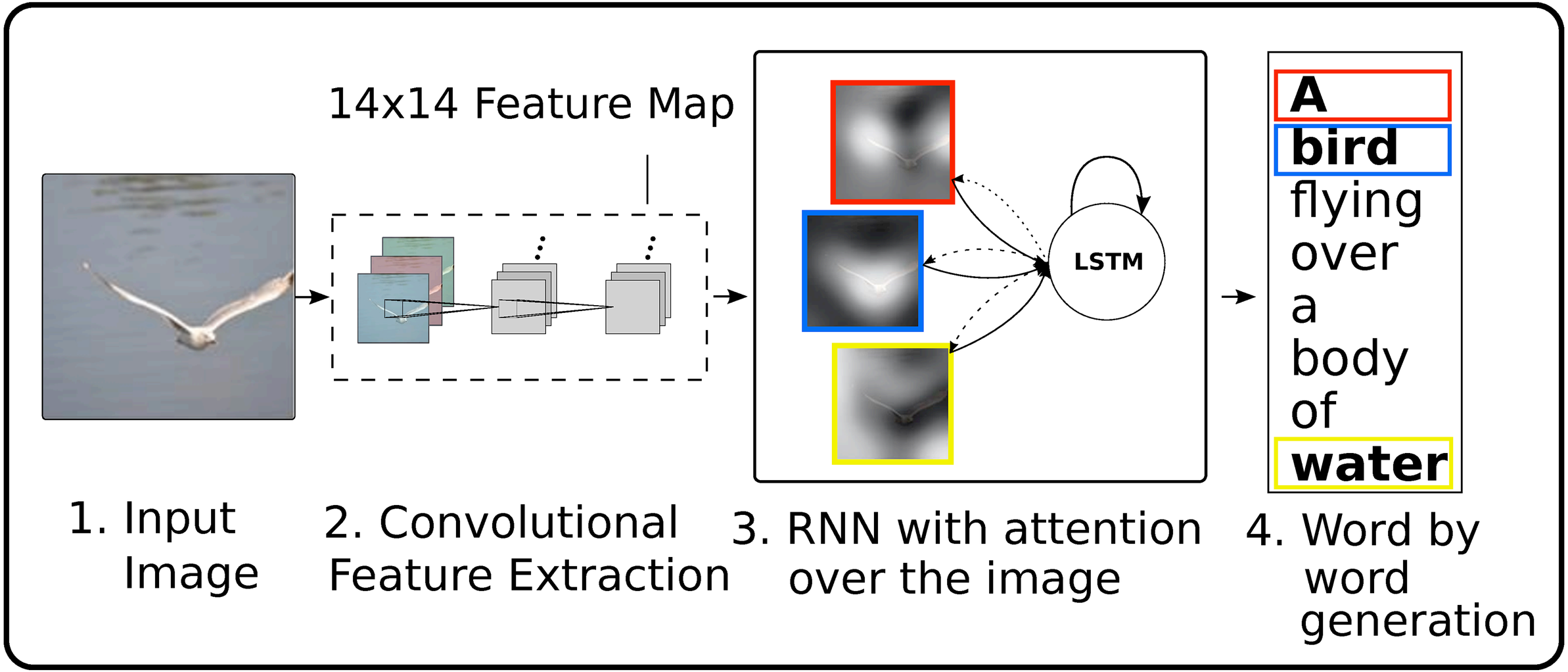
- Attention plays a similar role to word alignments



Attention in NMT

- Attention plays a similar role to word alignments
- It can also be seen as a form of pooling
- Attention solves the "information bottleneck" problem
  - It allows decoder to "look directly" at the source sequence
  - It also helps with the vanishing gradient problem
- Provides some interpretability

Attention in Image Captioning



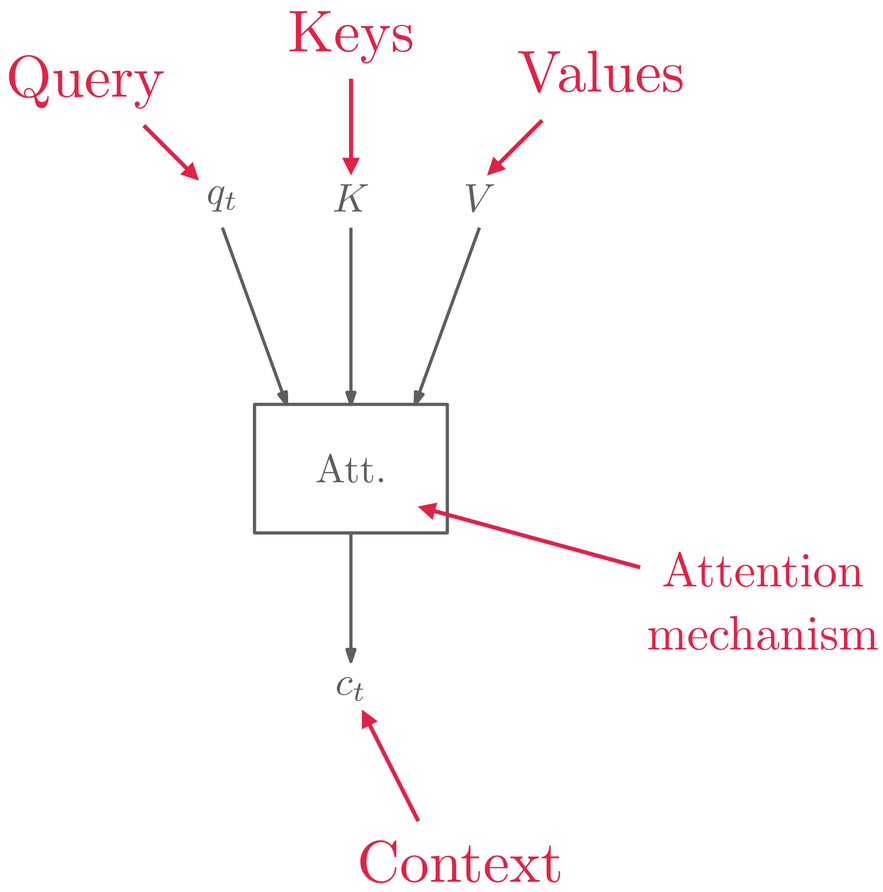
Attention in Image Captioning

Going even further...

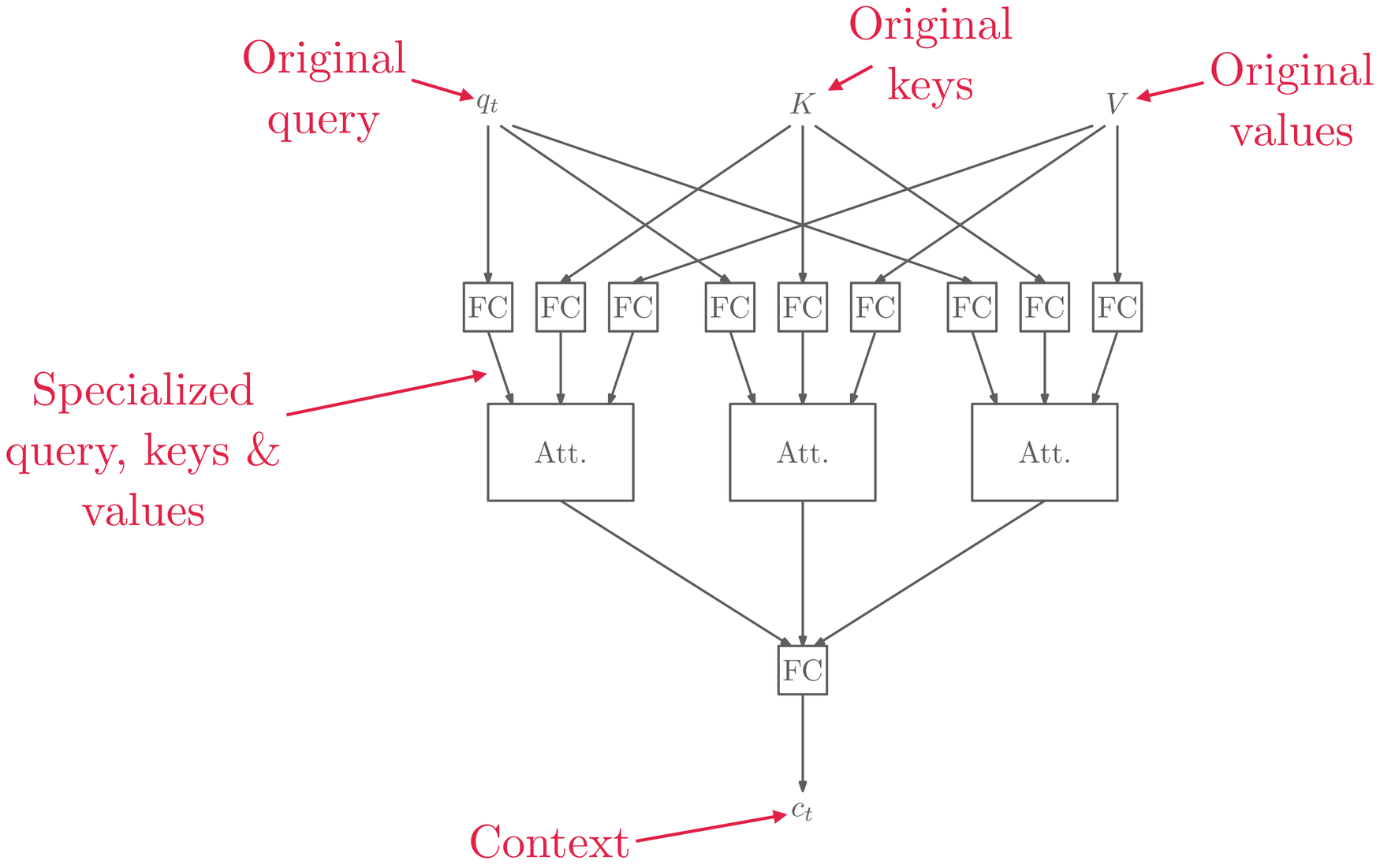
Multi-head attention

- We can include multiple "attention channels" by allowing the network to learn different ways to attend to the encoder states
  - Query, keys and values are transformed through fully connected layers
  - Attention applied to the outputs of FC layers—each a specialized context vector
  - Different context vectors are then merged through a fully connected layer

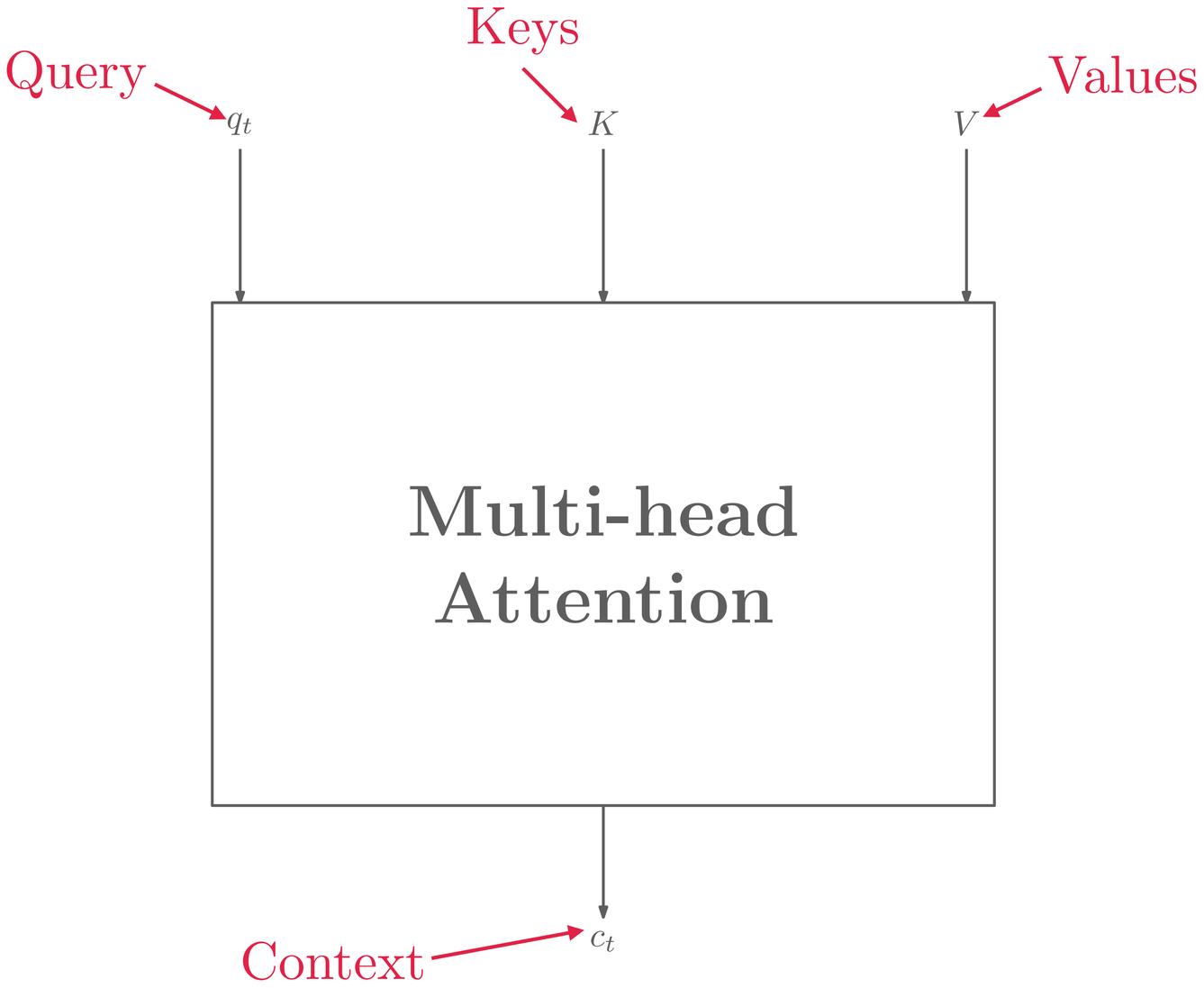
Single head attention



Multi-head attention



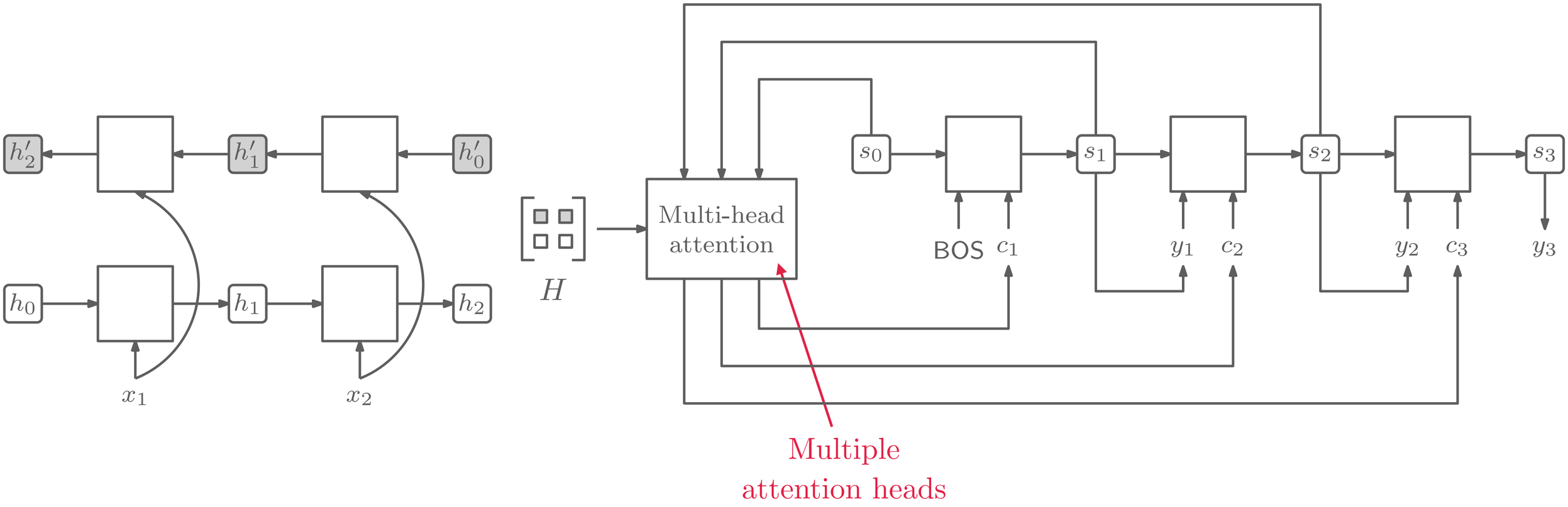
Multi-head attention



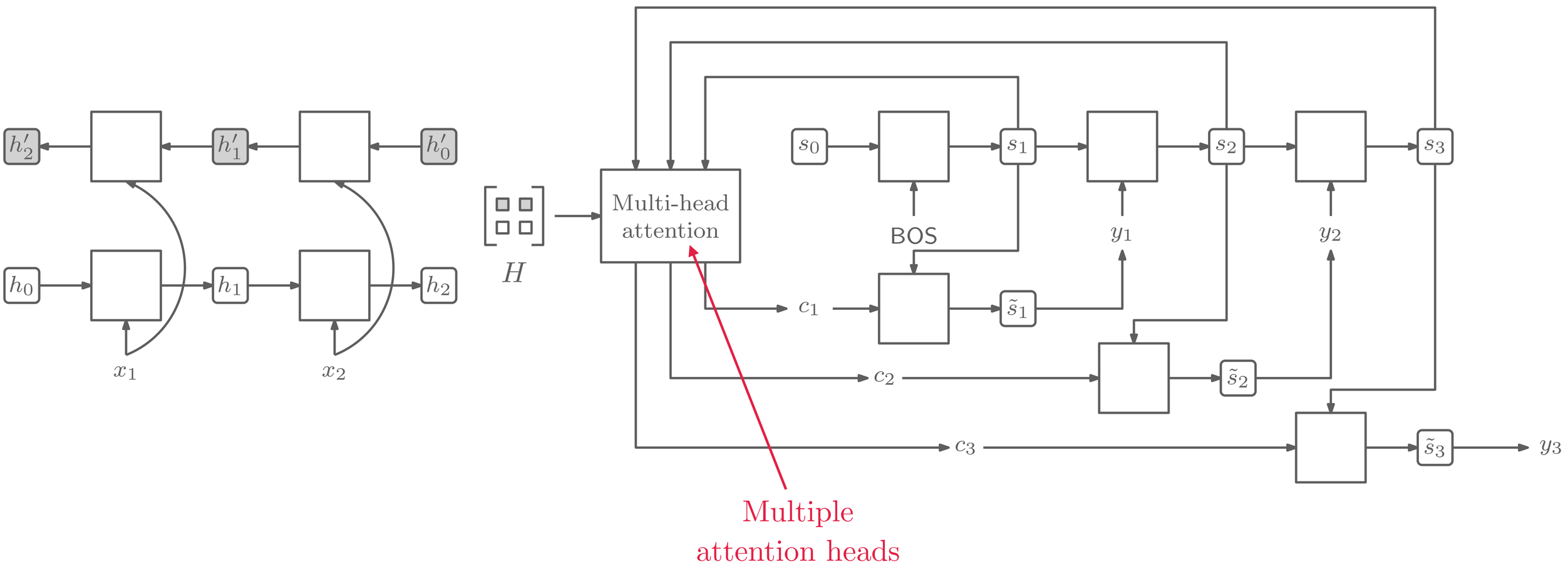
Multi-head attention

- Multi-head attention enables the decoder to capture dependencies of different ranges
- Plays a role similar to channels in CNNs (where each channel implements a specific filter)

Multi-head attention



Multi-head attention



Going even... even further...

- So far we have considered contextual attention:
  - The decoder attends to the encoder states (input context)
- If attention is so good, why not use it also in the encoder?
  - We have the encoder attend to its own states!

Self-attention

Going even... even further...

Self-attention

- Given a sequence of inputs

$$x_1, \ldots, x_T$$

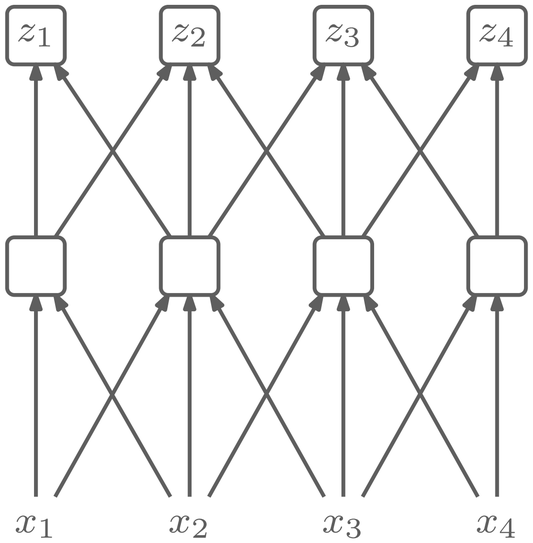
we build a sequence of representations

Play the role of encoder hidden states

$$z_1, \ldots, z_T$$

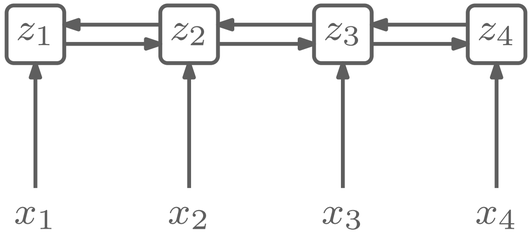
by attending to the different elements in the input sequence

Self-attention



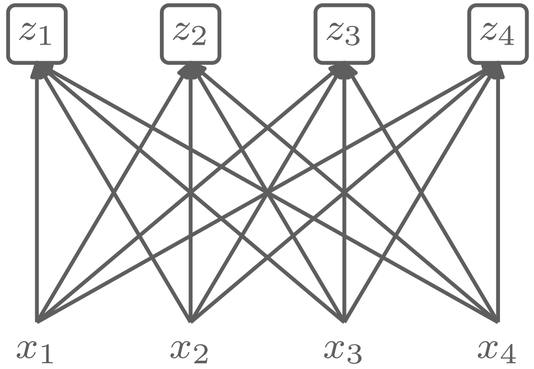
CNN

(hierarchical flow of information)



RNN

(sequential flow of information)



Self-attention

(parallel flow of information)

Implementing self-attention

Self-attention

- Queries, keys and values are built from the input sequence:

$$Q = W_Q X$$

$$K = W_K X$$

$$V = W_V X$$

Projection matrices

Implementing self-attention

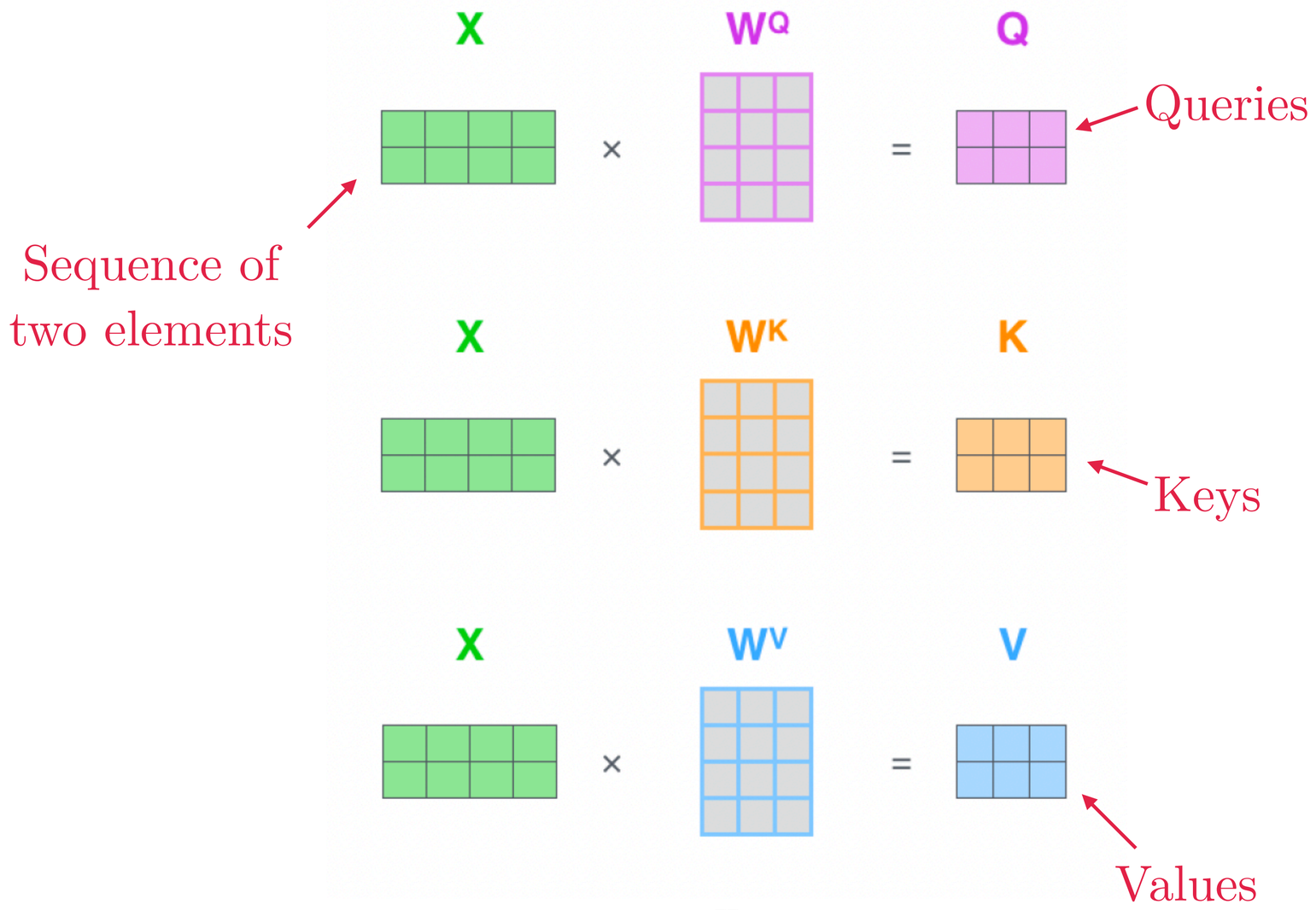
Self-attention

- We use scaled dot-product

$$Z = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_K}}\right)V$$

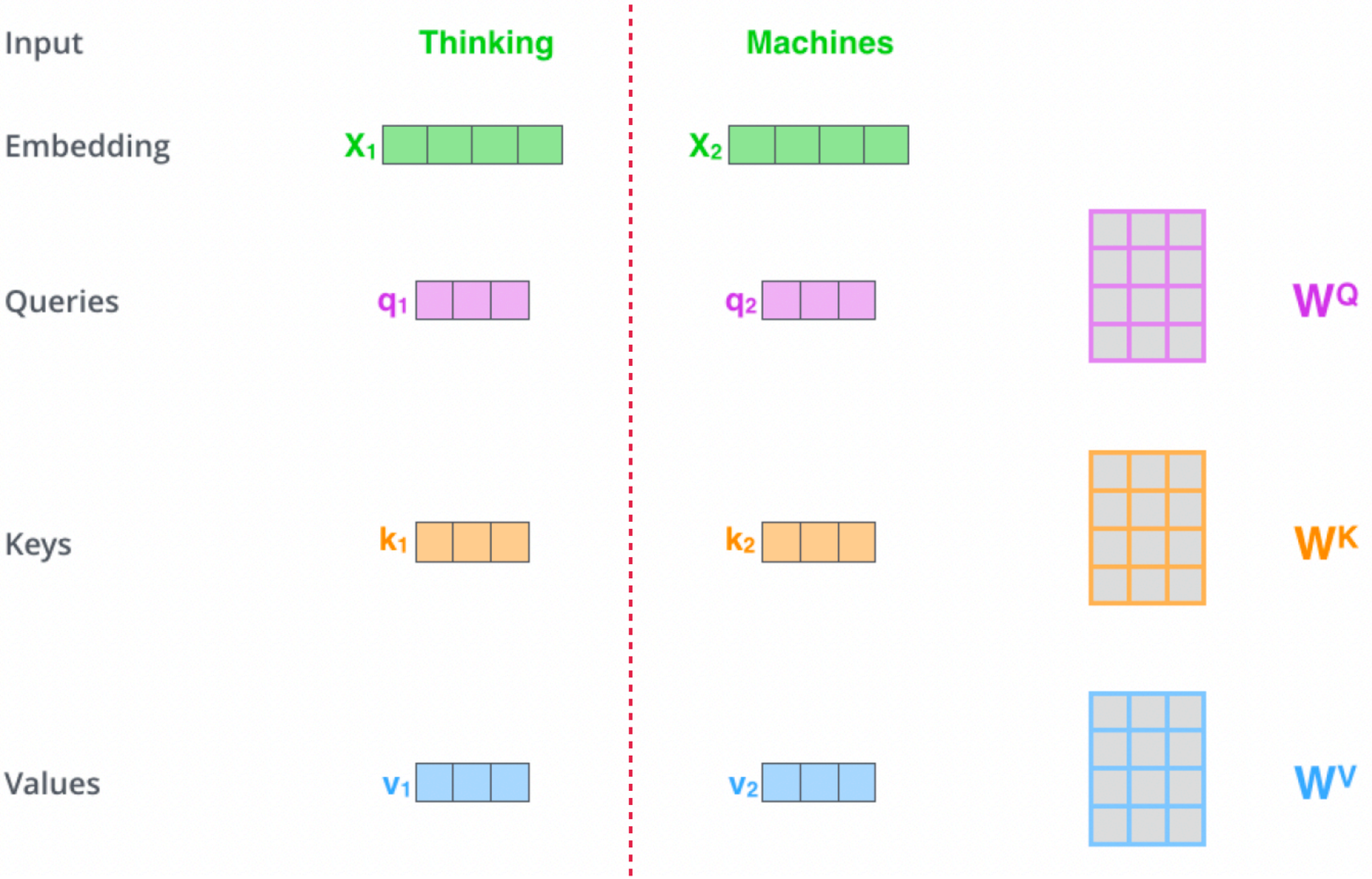
- We can parallelize the computation!

Implementing self-attention

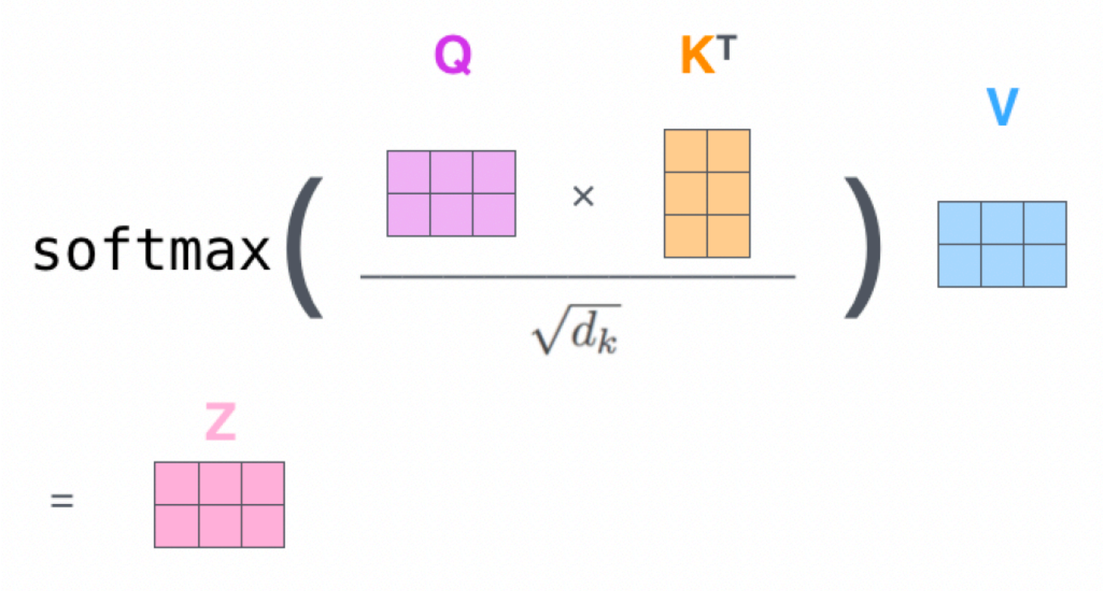


Implementing self-attention

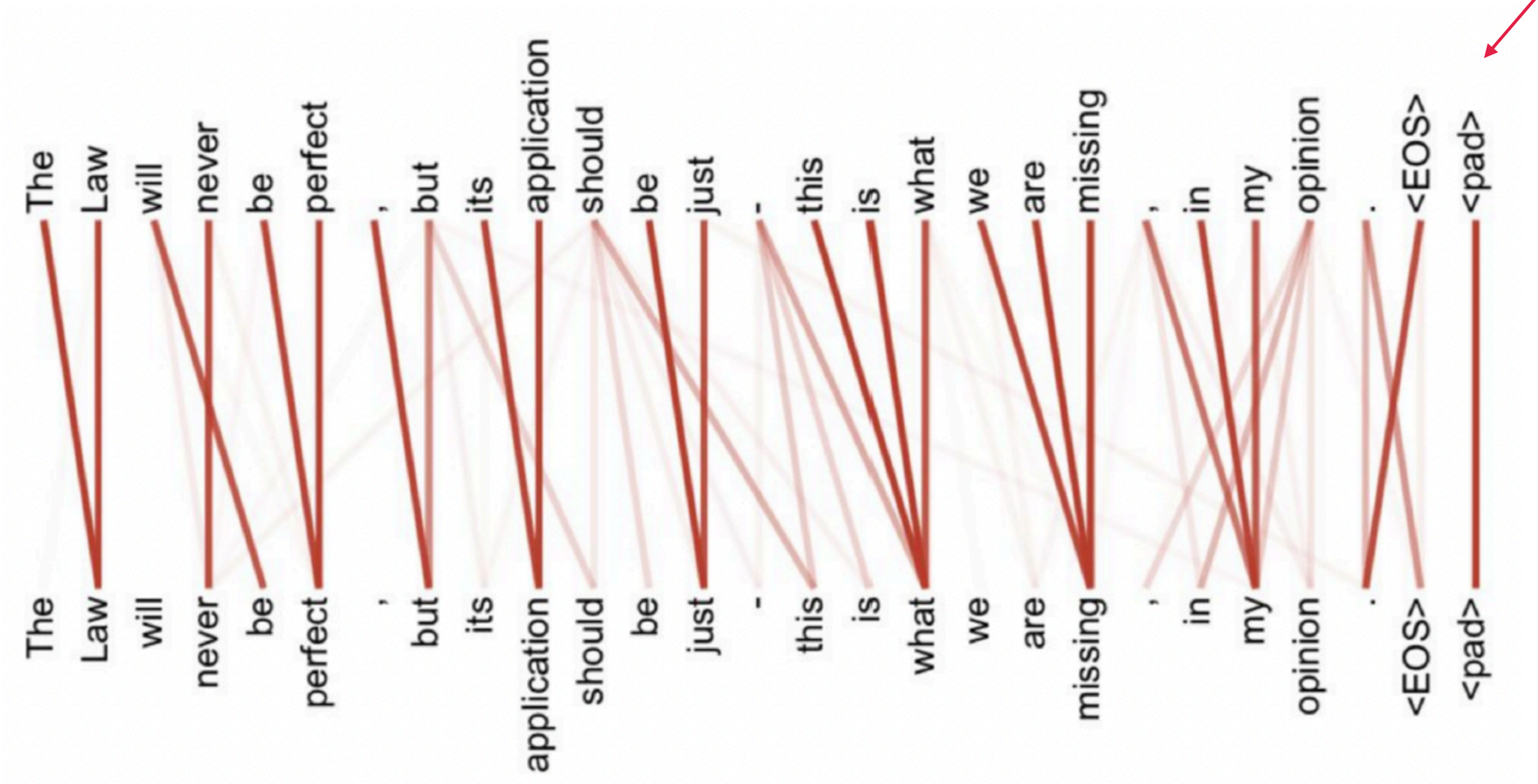
Computation can be parallelized



Implementing self-attention



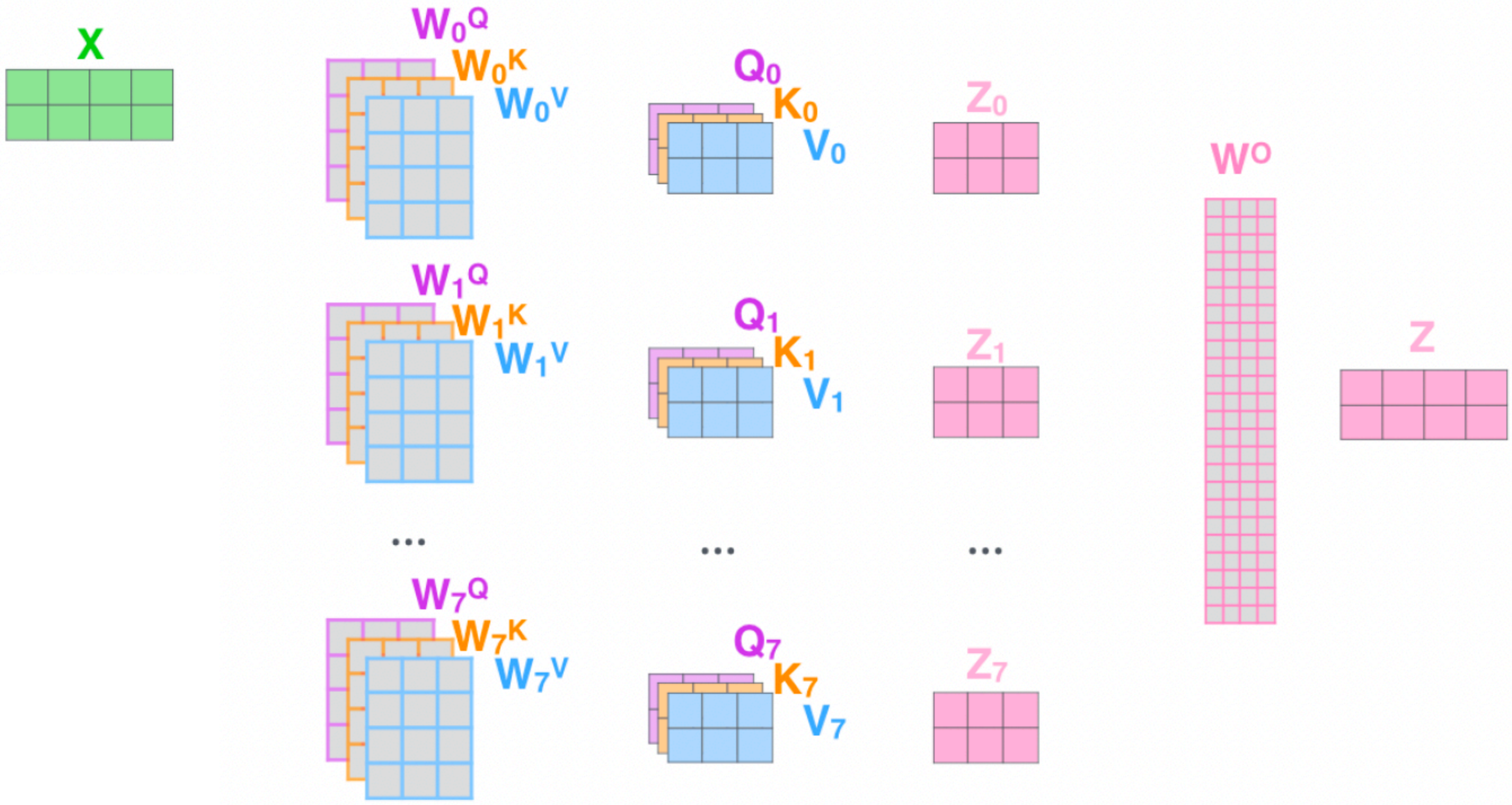
Example



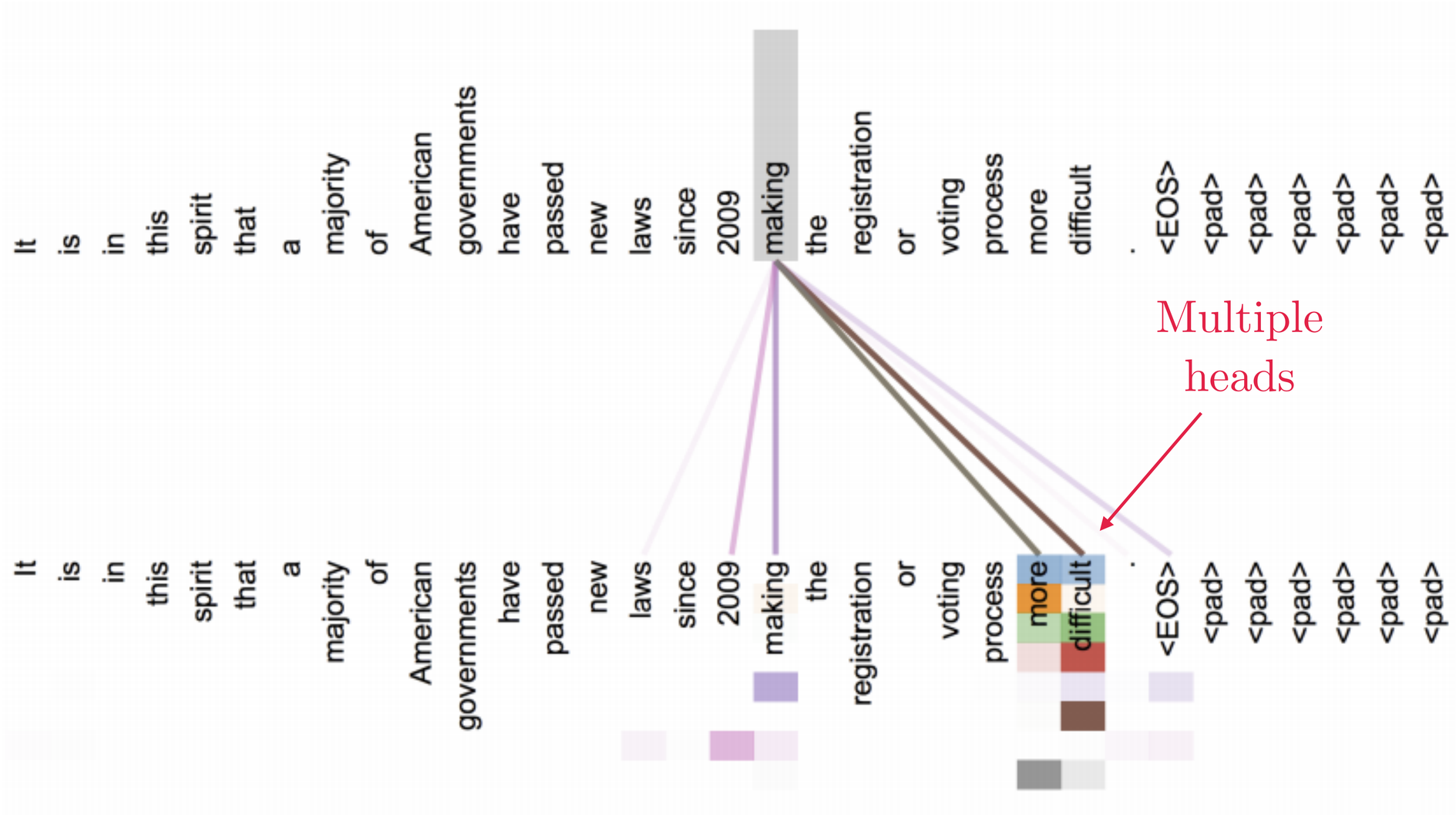
Multi-head self-attention

- We can also have self-attention with multiple heads
  - Each head has its own query, key and value projection matrices

Multi-head self-attention

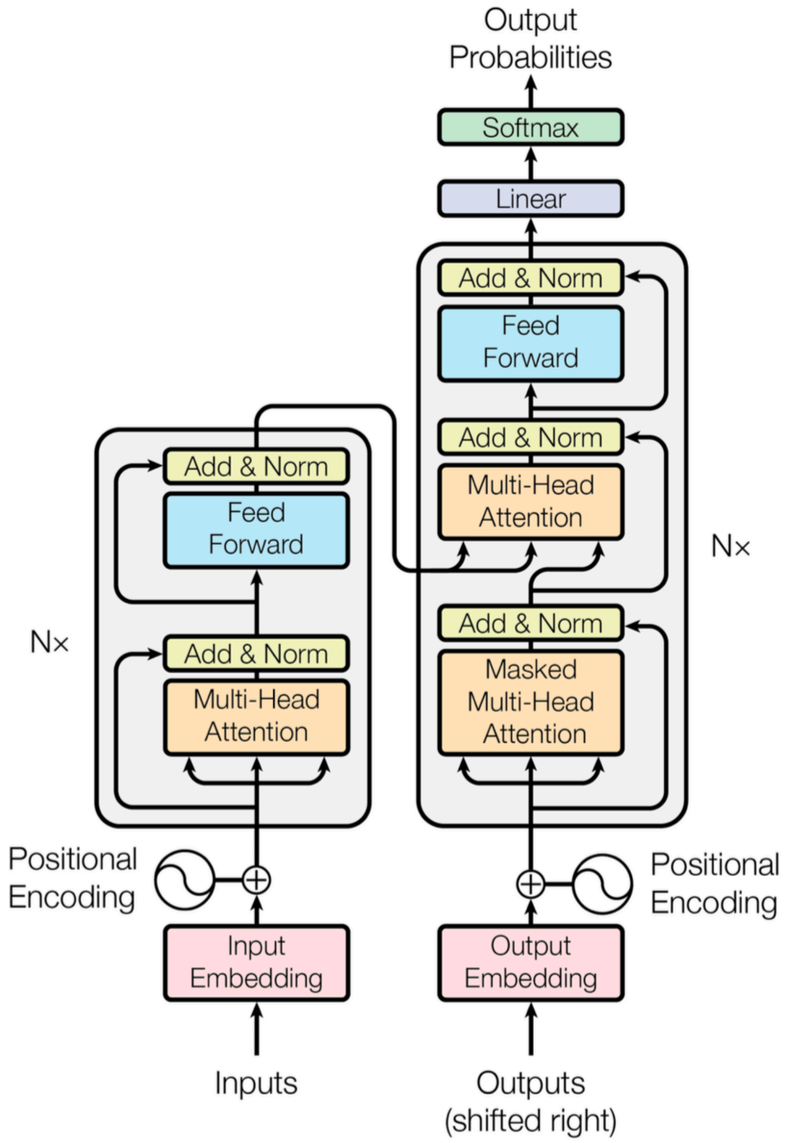


Example



Can we get rid of recurrence on the decoder, also?

Transformers



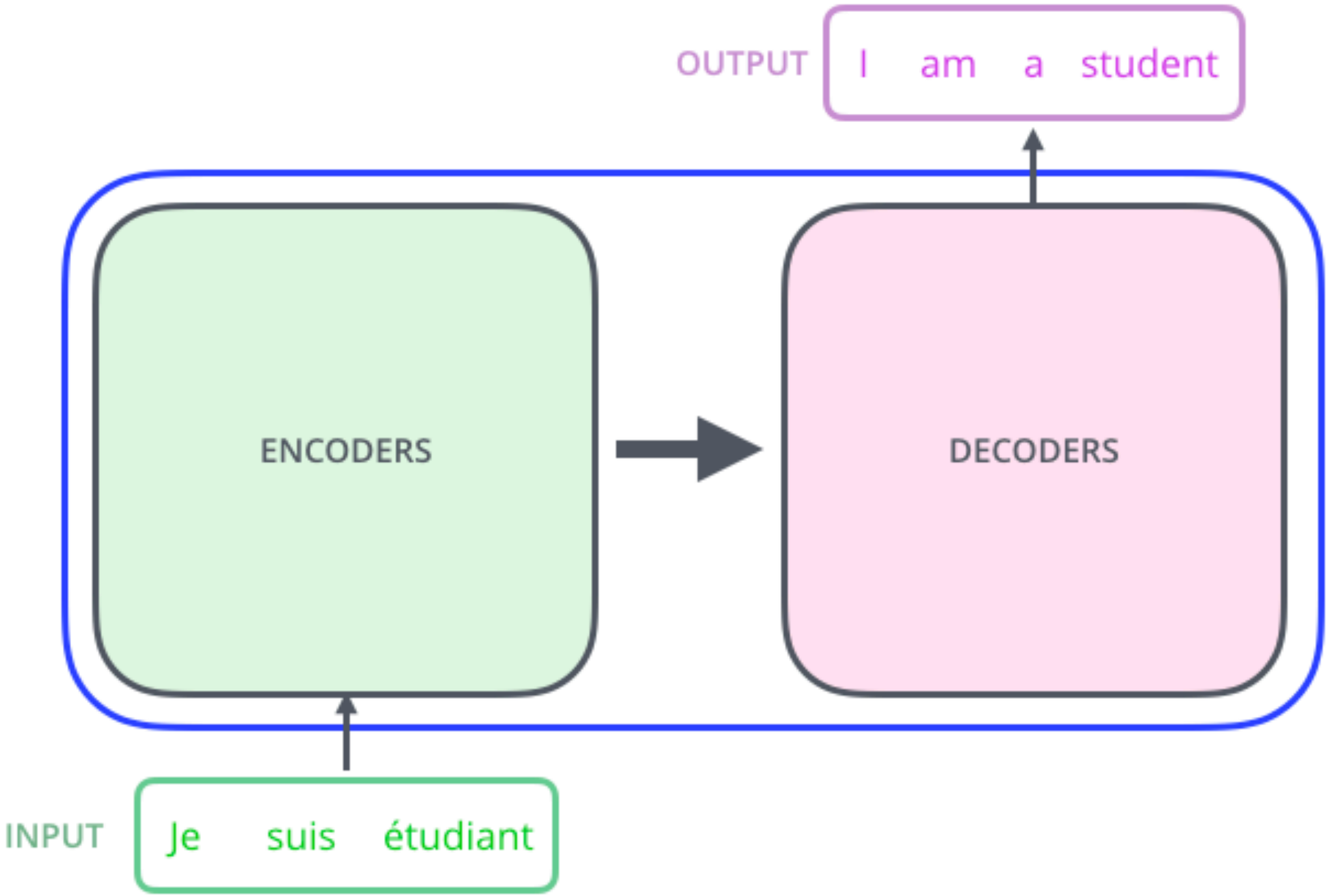
Transformers

- Idea:
  - Encoder learns a representation of input sequence using self-attention
  - Decoder generates an output sequence using both self-attention and contextual attention

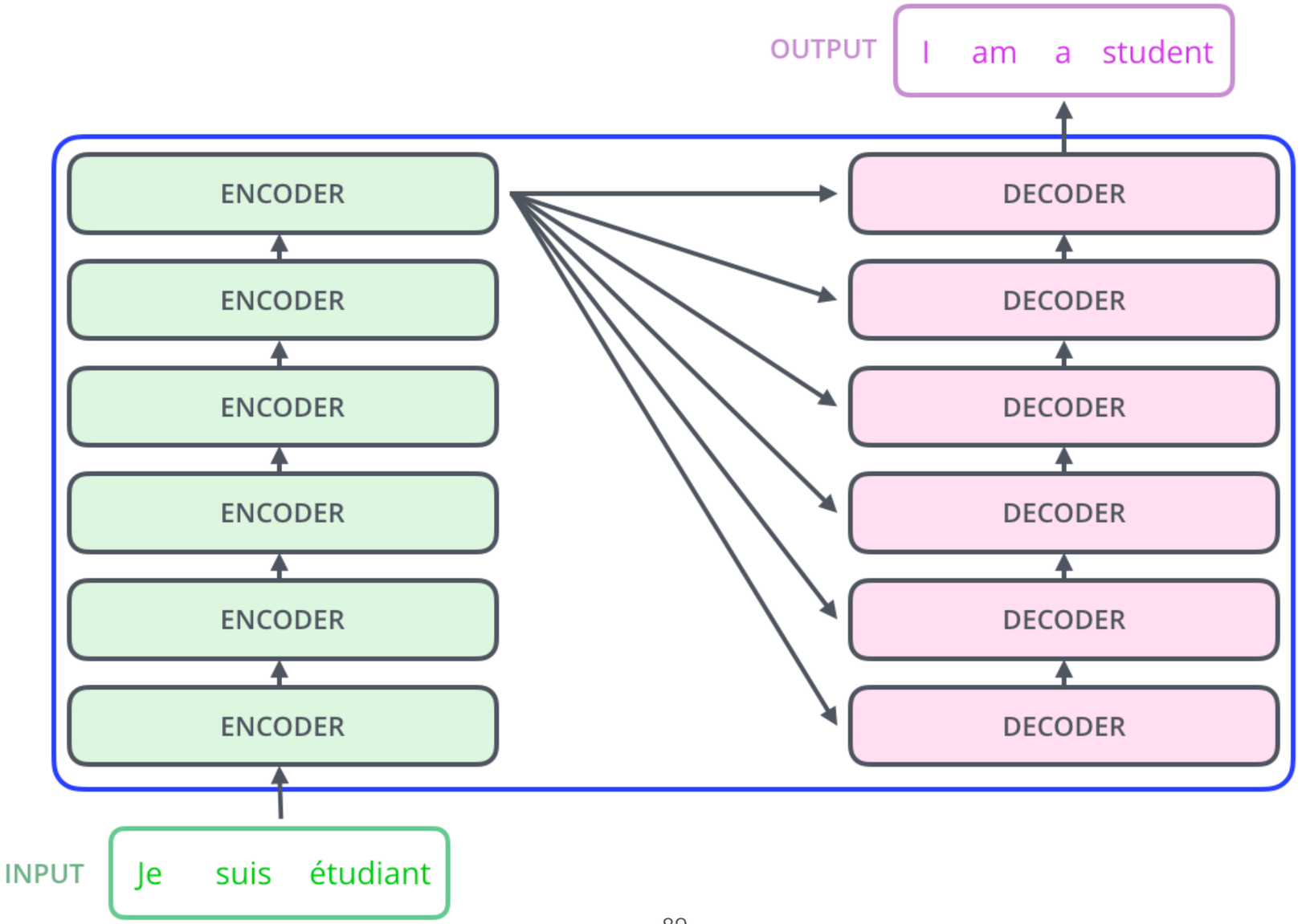
Transformers

Illustrations from http://jalammar.github.io/illustrated-transformer/

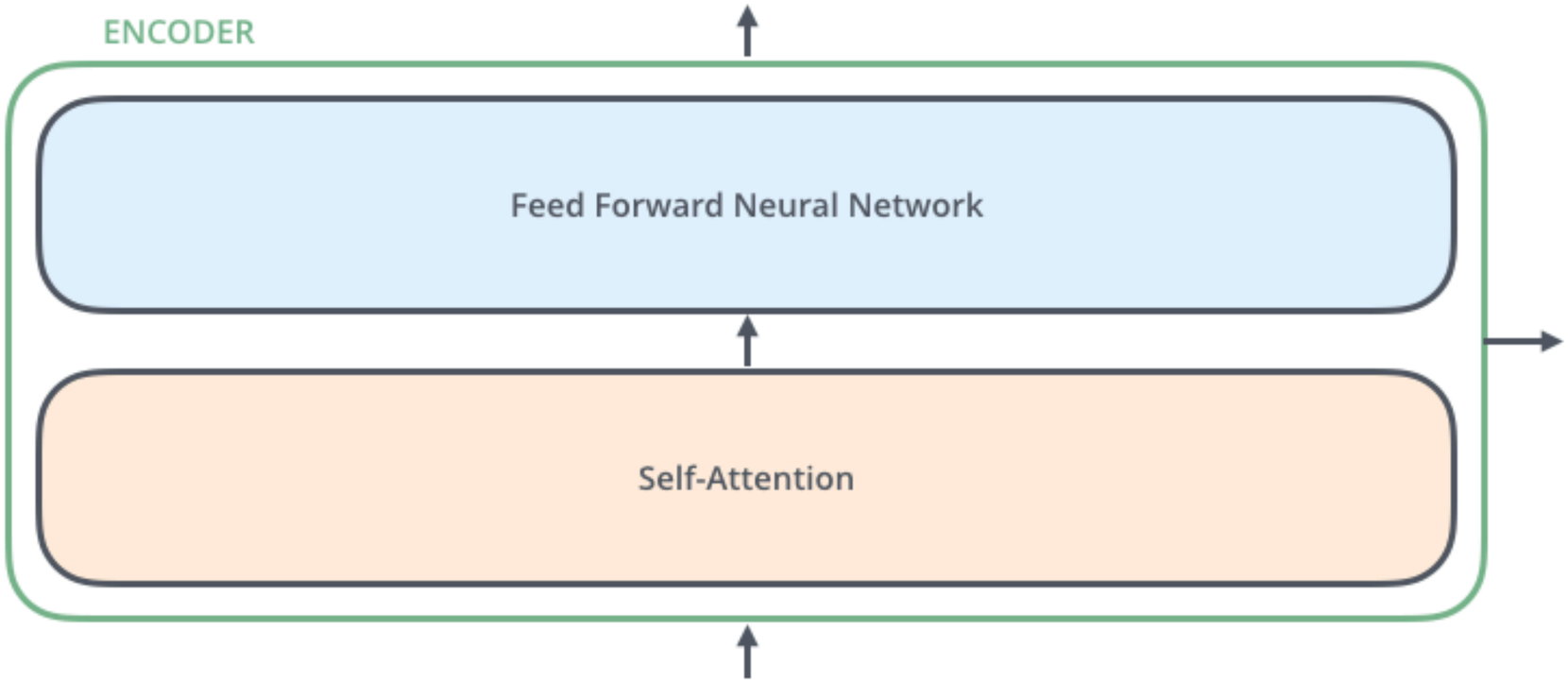
Transformers



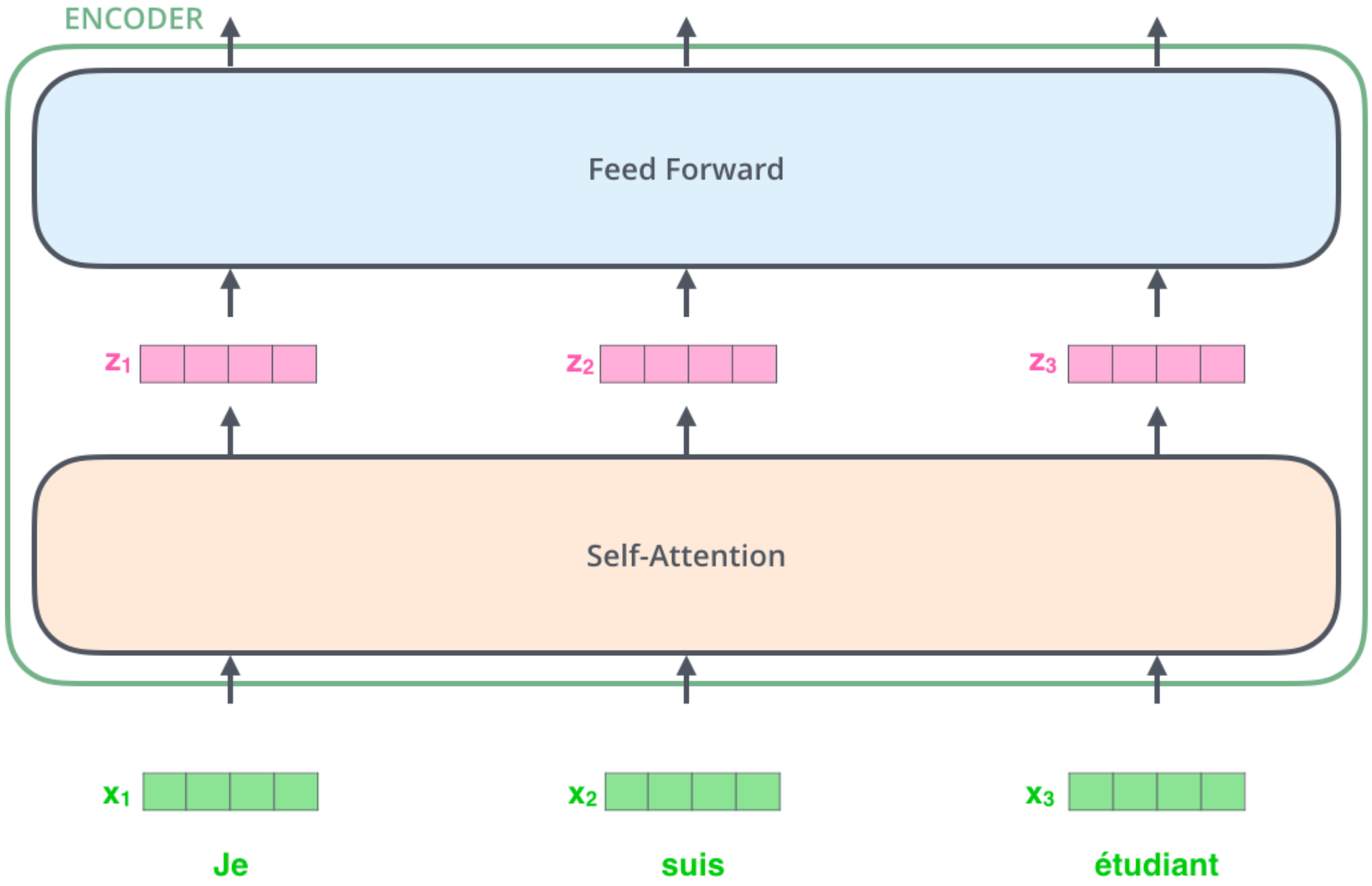
Transformers



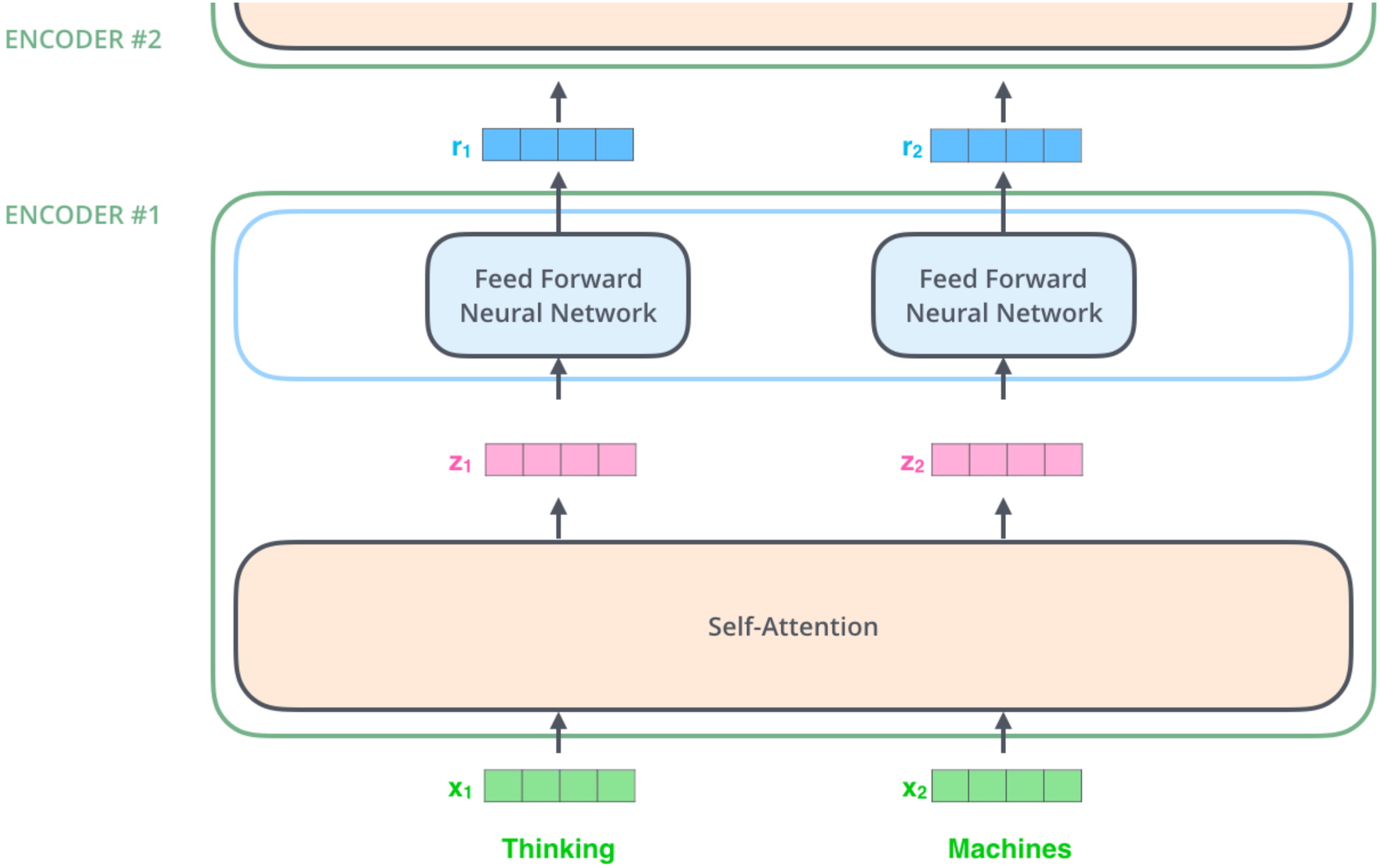
The encoder



The encoder


ENCODER
Feed Forward
z1
z2
z3
Self-Attention
x1
x2
x3
Je
suis
étudiant

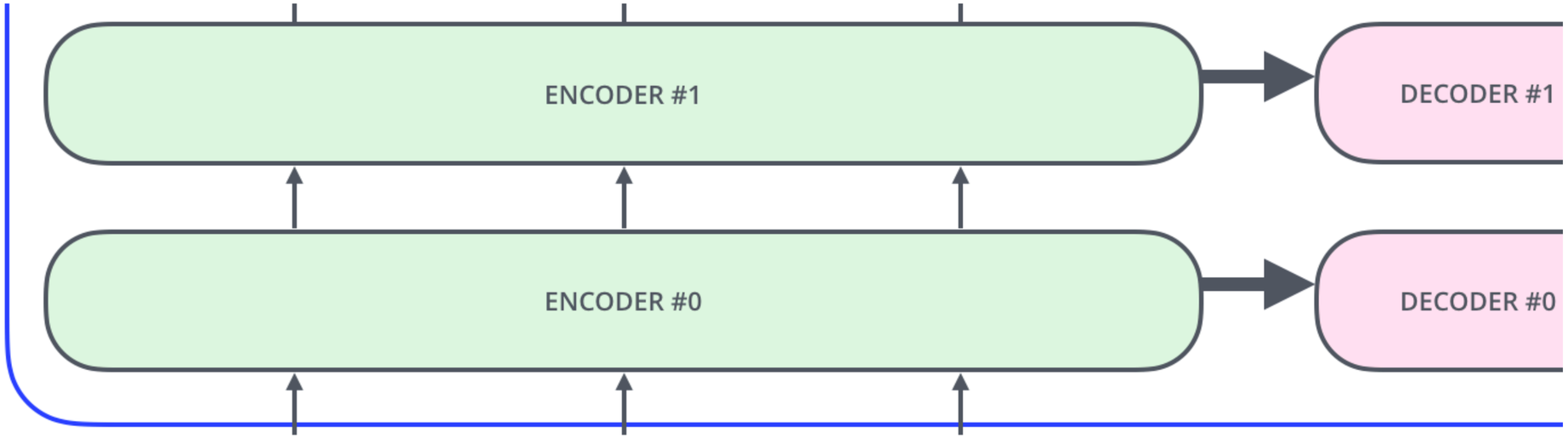
The encoder

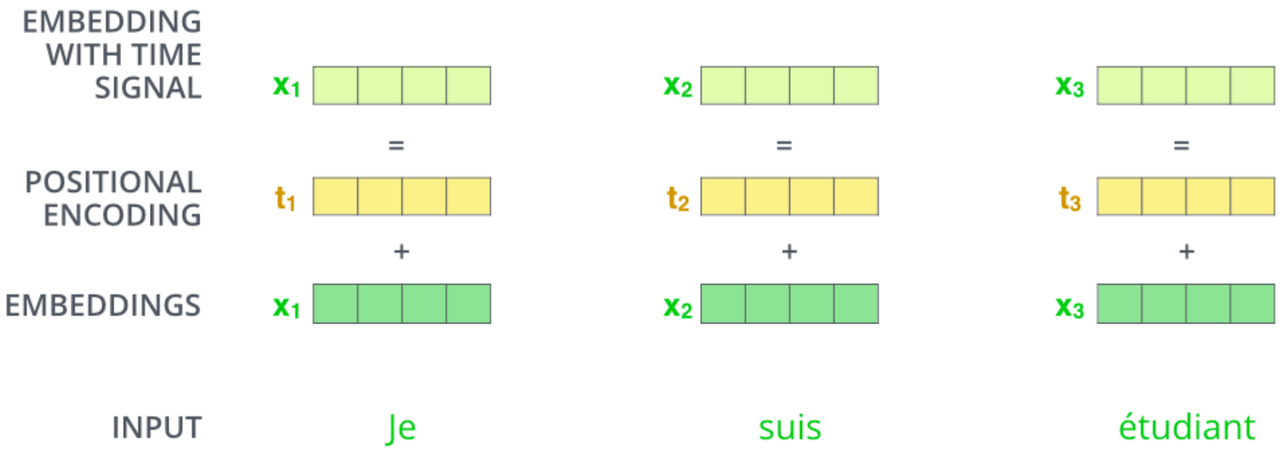

ENCODER #2
ENCODER #1
r1
r2
Feed Forward Neural Network
Feed Forward Neural Network
z1
z2
Self-Attention
x1
x2
Thinking
Machines

The encoder

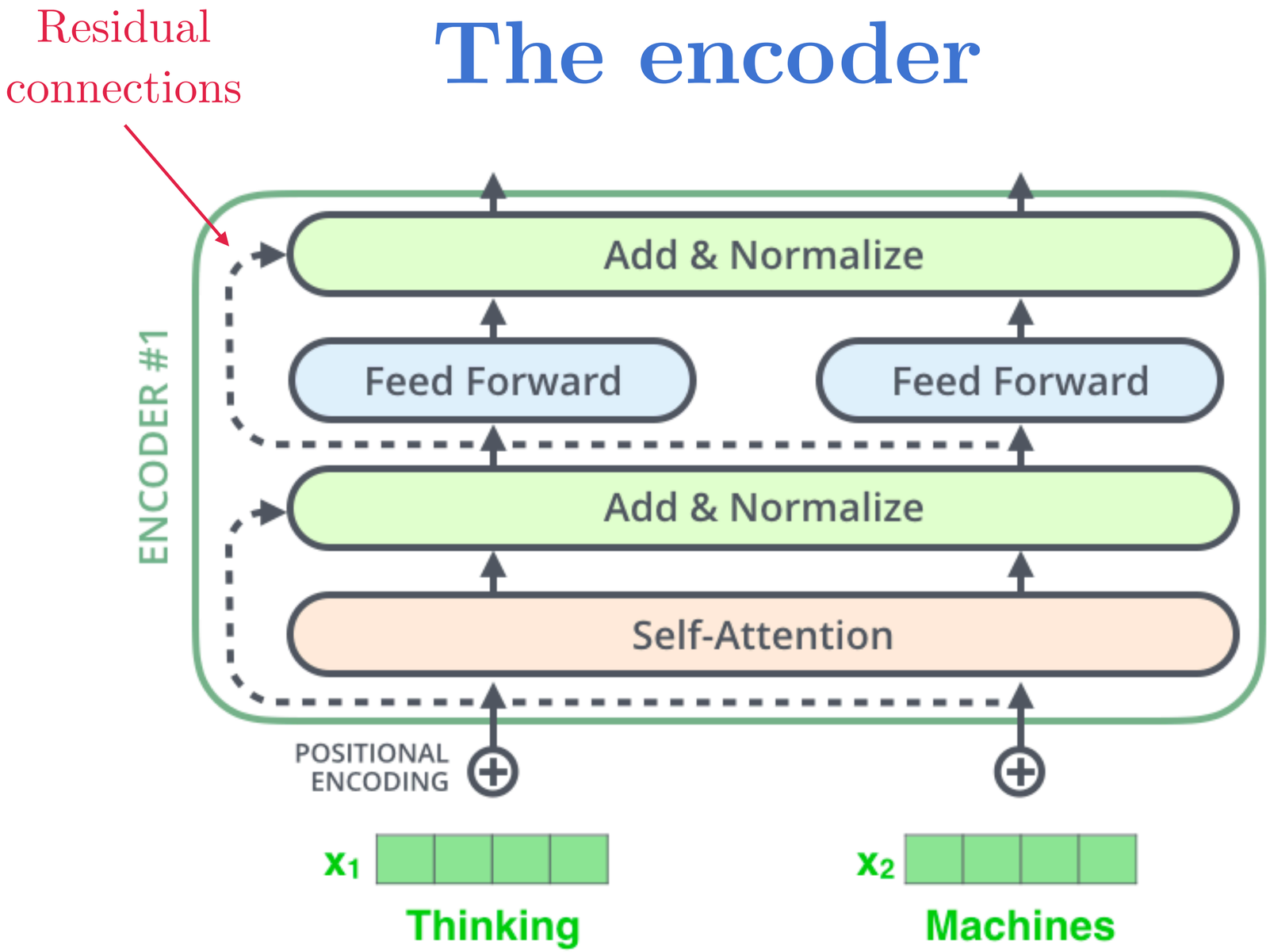
- Input tokens are processed in parallel
- This means that the order of the input tokens is lost
- Positional encoding enables the encoder to have different representations for the same word when it appears in different positions

The encoder

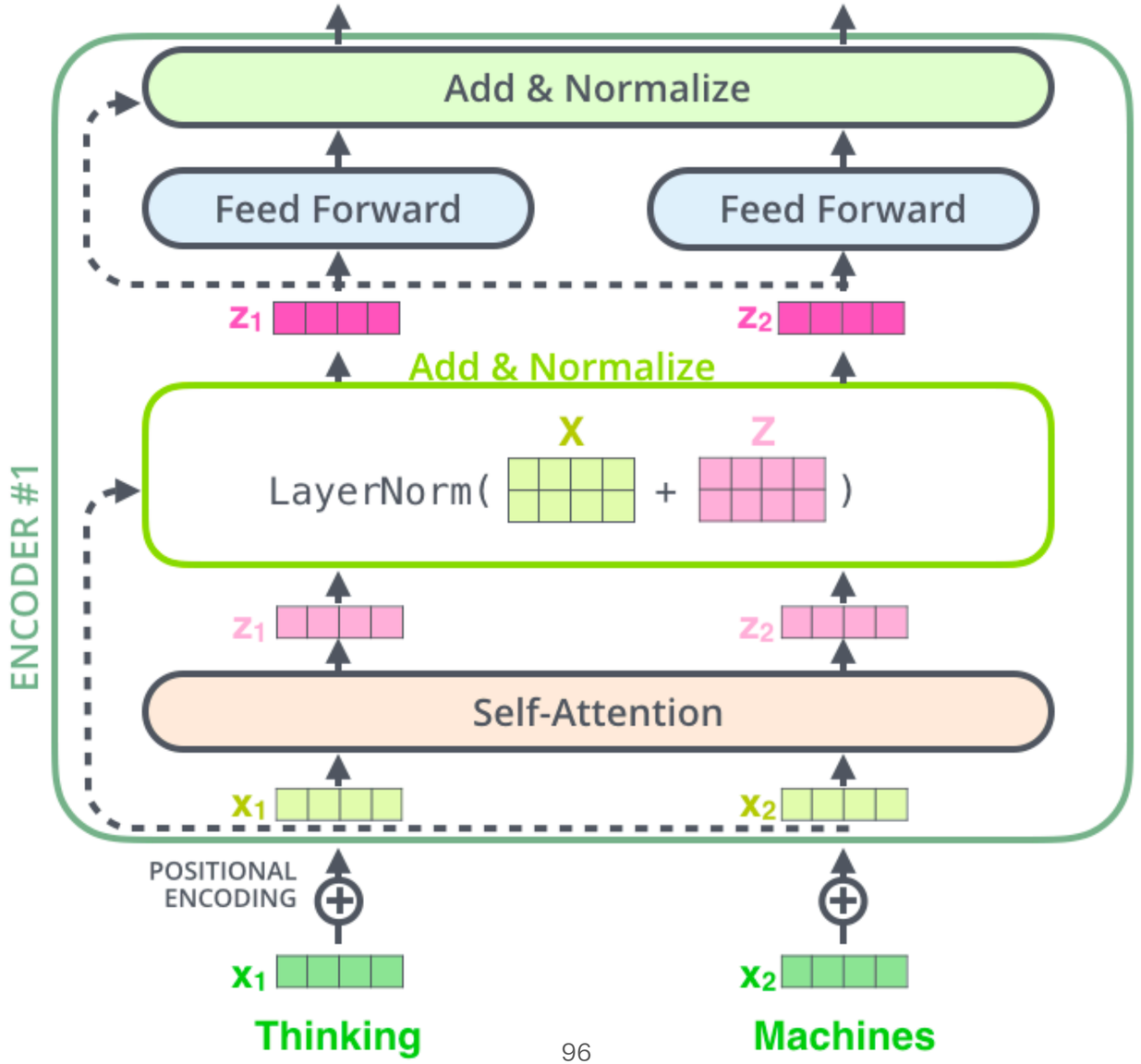




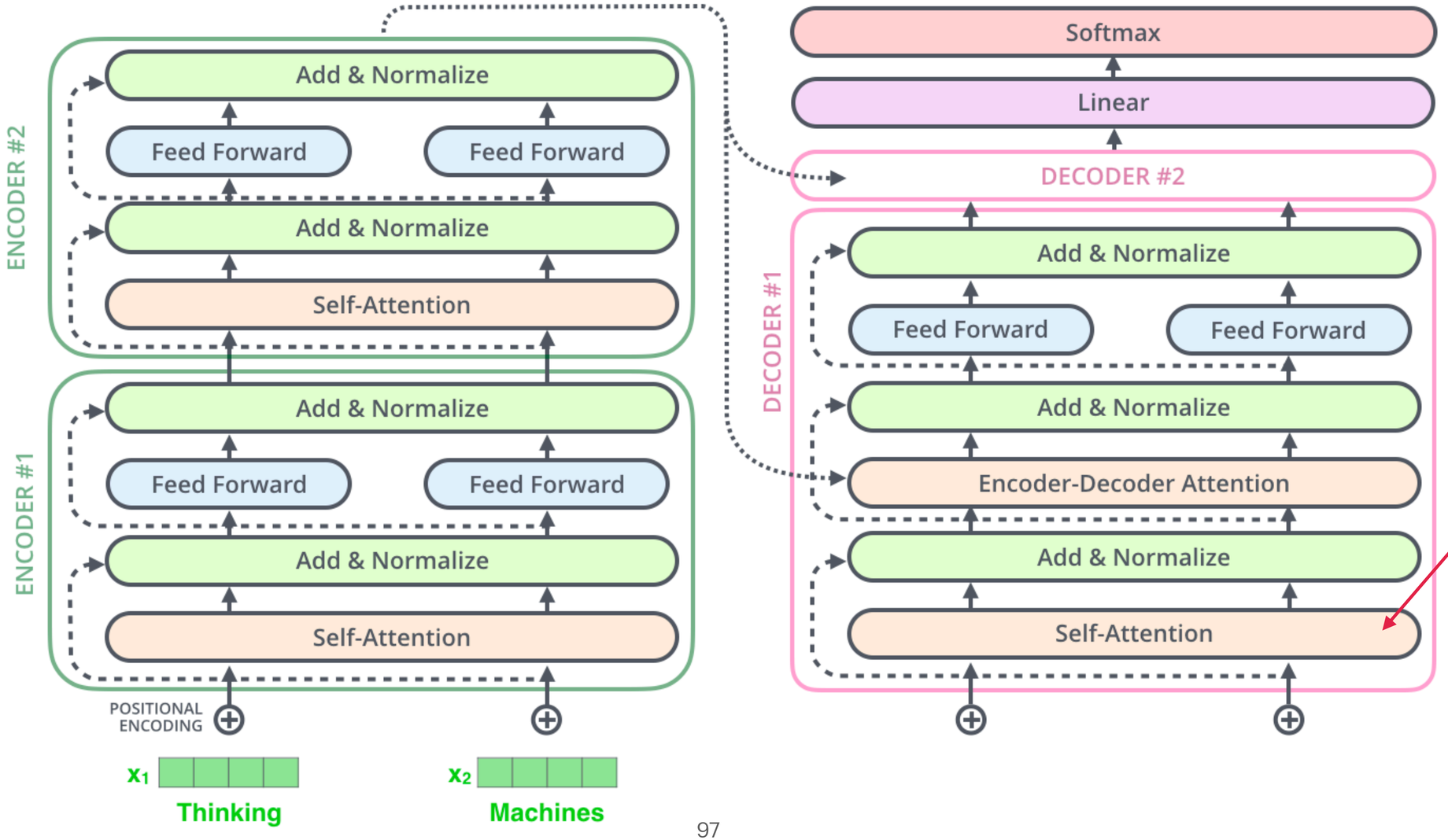
The encoder



The encoder

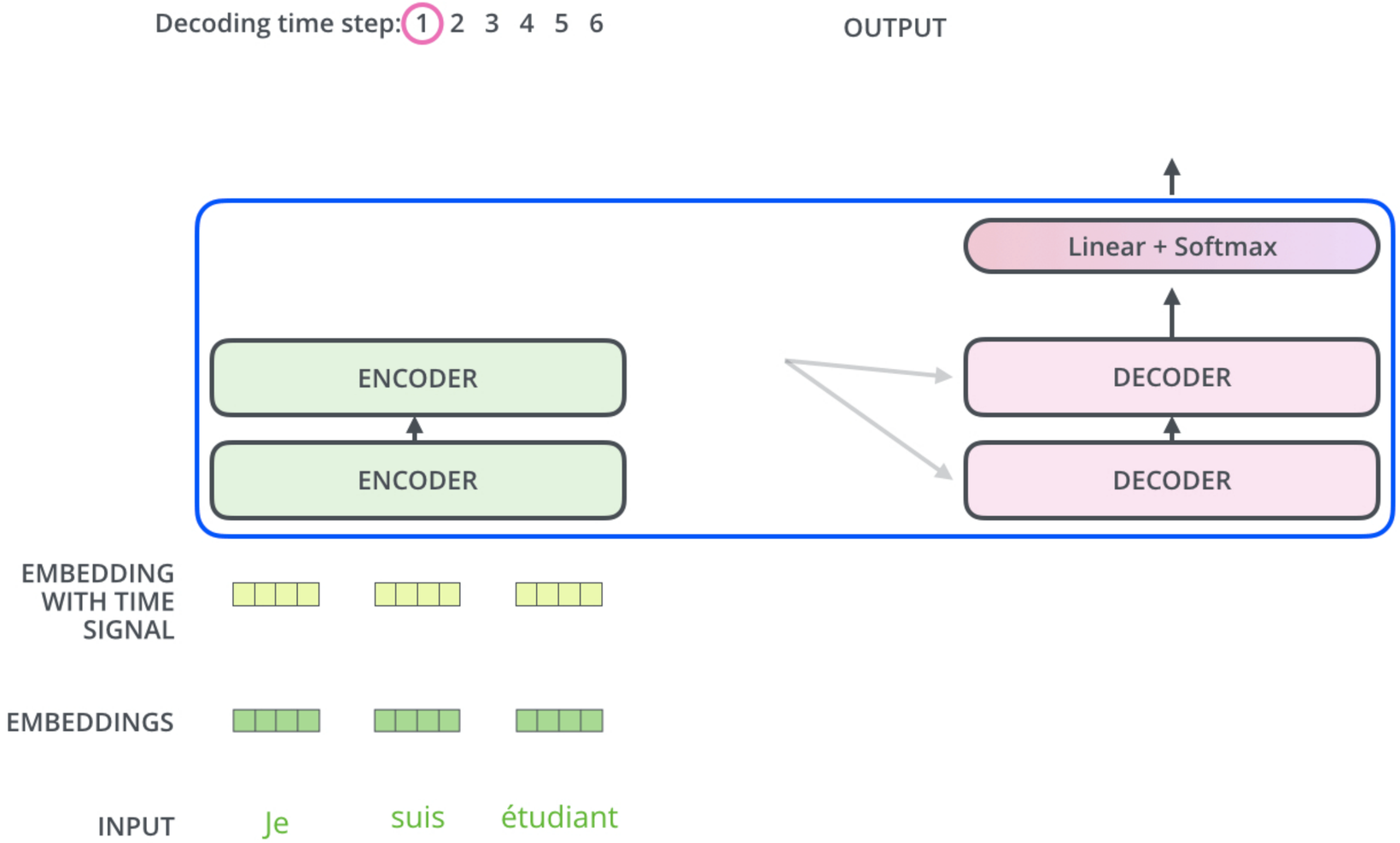


Bringing in the decoder...

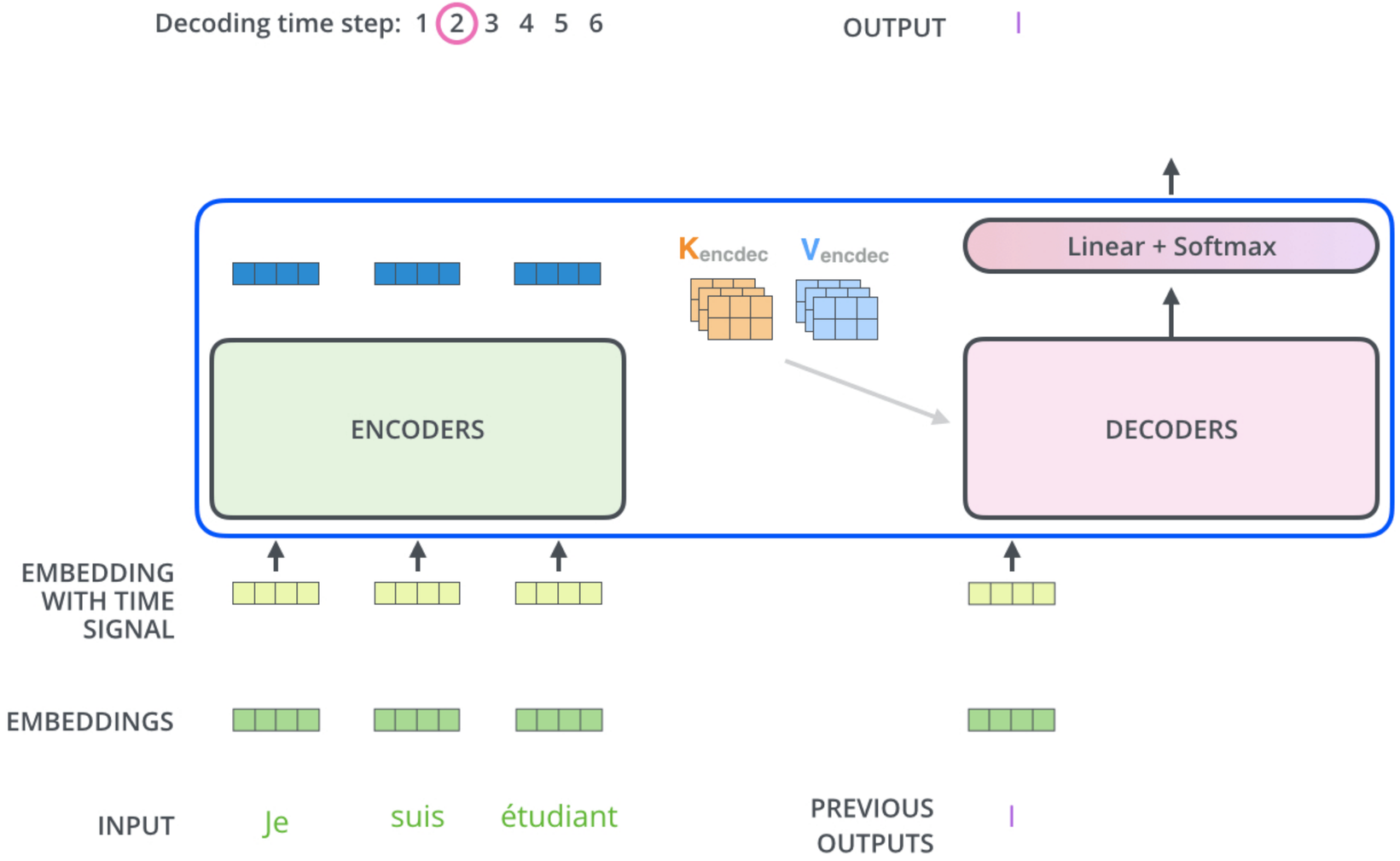


Causal masking during training

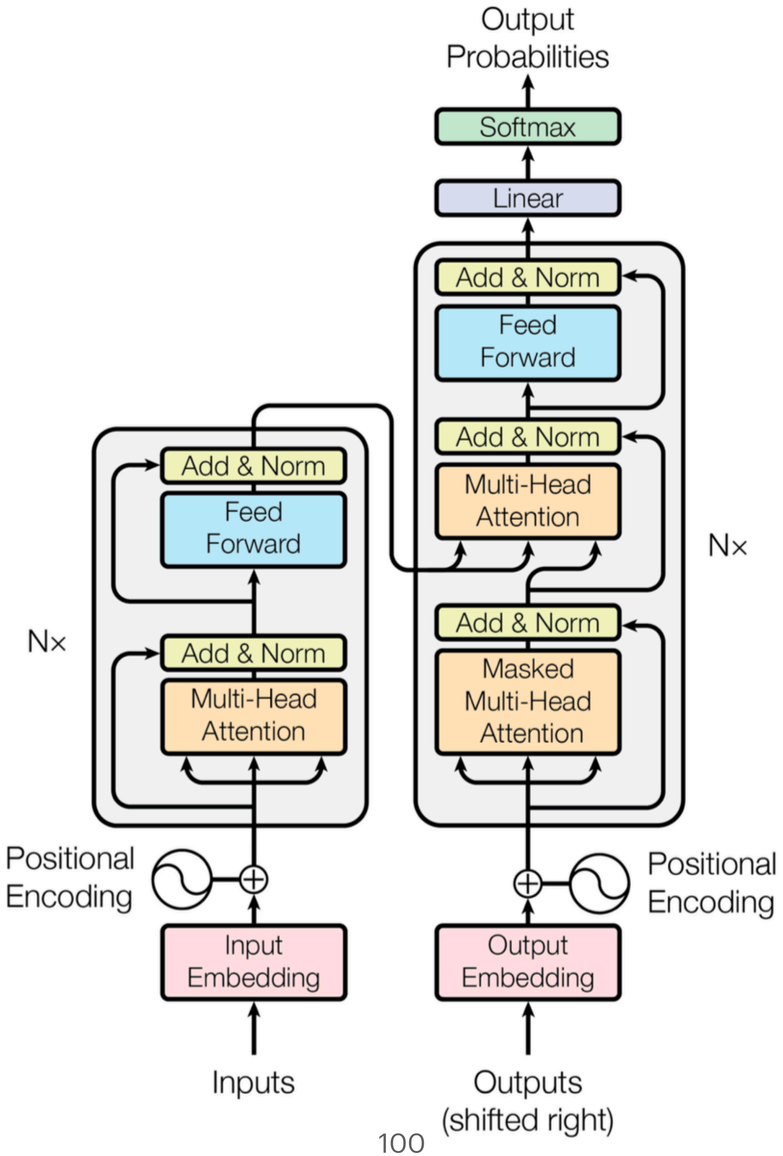
Everything together...



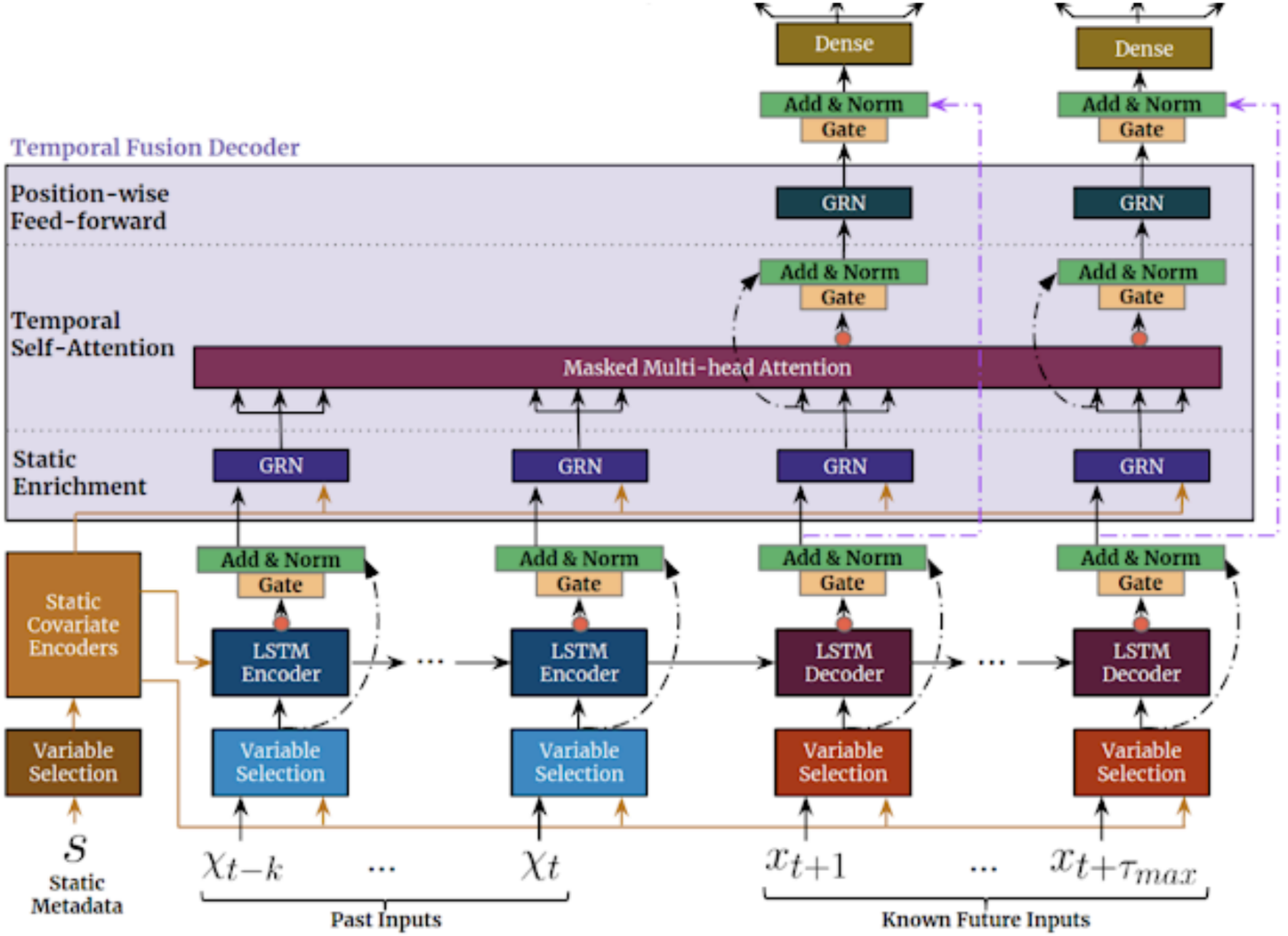
Everything together...



Back to the original diagram...

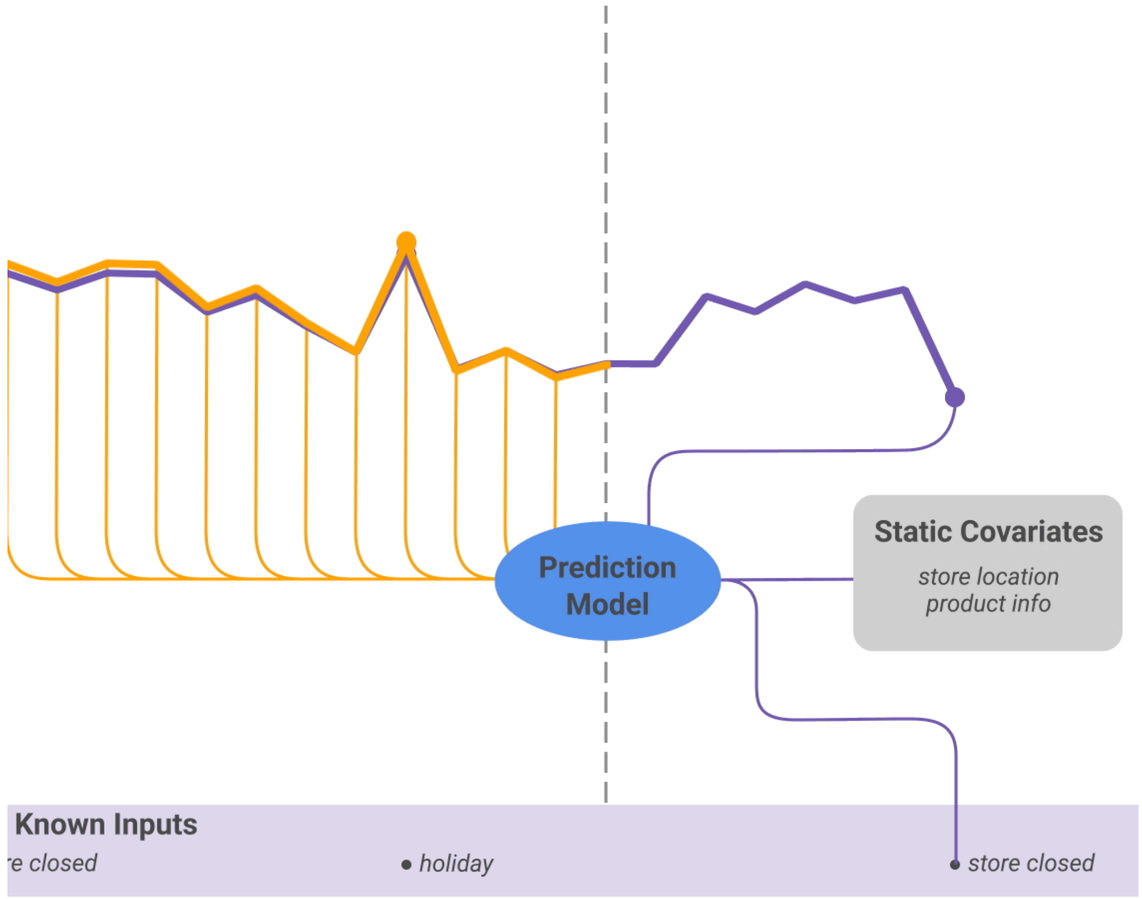


Multi-head attention for forecasting



Multi-head attention for forecasting

Temporal Fusion Transformers for interpretable multi-horizon time series forecasting



Multi-head attention for forecasting

Temporal Fusion Transformers for interpretable multi-horizon time series forecasting

Model	Electricity	Traffic	Volatility	Retail
ARIMA	0.154 (+180%)	0.223 (+135%)	-	-
ETS	0.102 (+85%)	0.236 (+148%)	-	-
DeepAR	0.075 (+36%)	0.161 (+69%)	0.050 (+28%)	0.574 (+62%)
Seq2Seq	0.067 (+22%)	0.105 (+11%)	0.042 (+7%)	0.411 (+16%)
MQRNN	0.077 (+40%)	0.117 (+23%)	0.042 (+7%)	0.379 (+7%)
TFT	**0.055**	**0.095**	**0.039**	**0.354**

P50 ***quantile losses*** *(lower is better) for TFT vs. alternative models.*

Resources:

- blogpost
- paper
- tutorial 1, tutorial 2

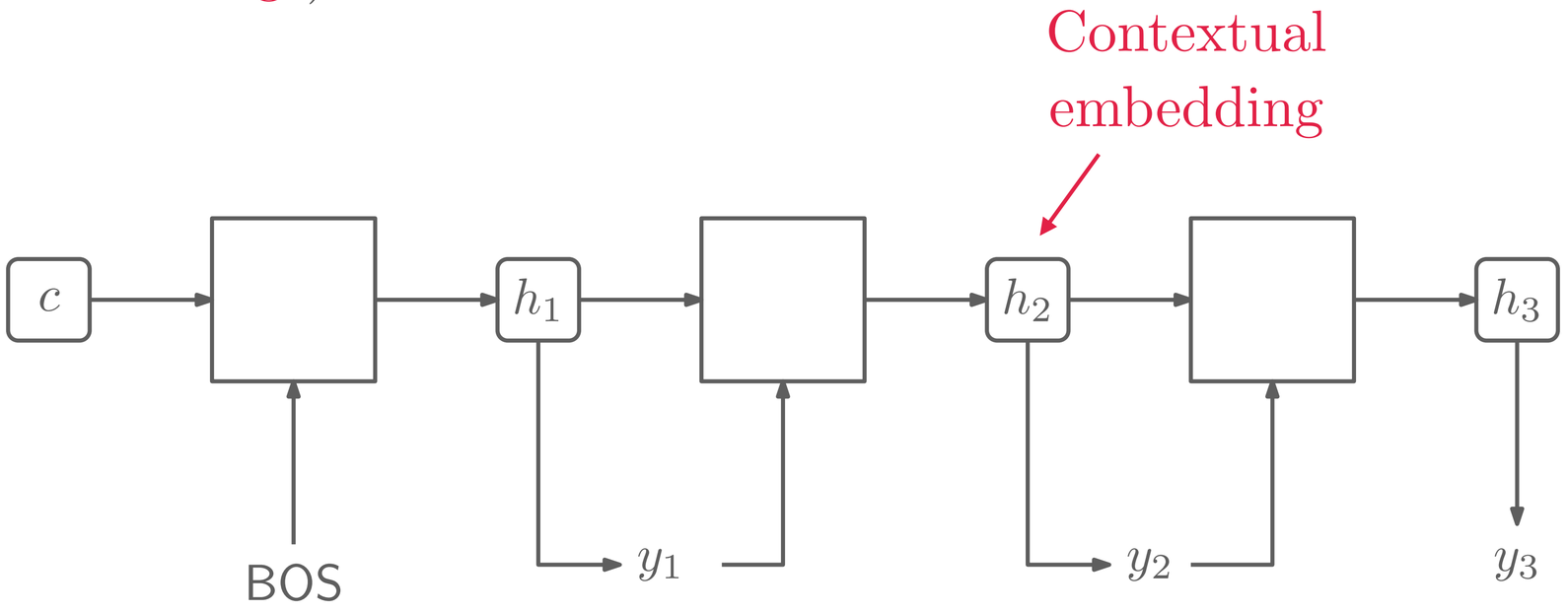
Large pre-trained models

Word embeddings

- Word embeddings provide static representations of words as vectors
- However, the same word can have different meanings depending on context:
  - "*There are two flies on my soup.*"
  - "*I don't understand how an airplane flies.*"
  - "*Today, on my way to work, I saw a bear!*"
  - "*Oh, I can't bear to watch!*"

Contextualized embeddings

- What if we use a learned language model to provide context-dependent representations (contextual embeddings)?



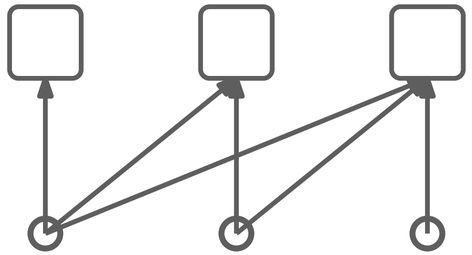
Contextualized embeddings

- What if we use a learned language model to provide context-dependent representations (contextual embeddings)?
  - In 2018, a model explored this idea, using a large bi-directional language model
  - The embeddings from this model led to significant better performance in downstream tasks
  - The model was called Embeddings from Language Models (ELMo)

Pre-trained models

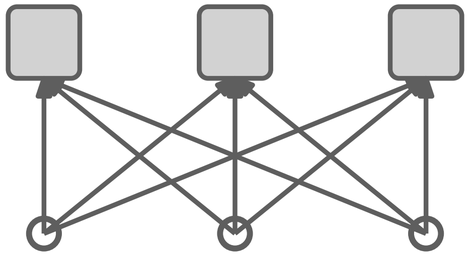
- Language models can be learned in a self-supervised manner—they require only a dataset of sentences
  - Large language models can be pre-trained on very large datasets (e.g., Wikipedia)
  - The representations learned can then be used in subsequent tasks with minor adjustments (fine-tuning)

Architectures for pre-trained models



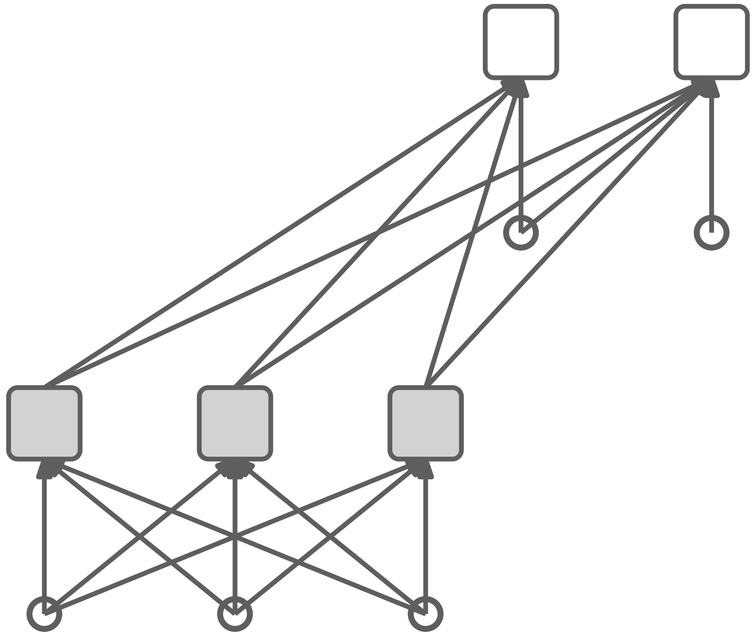
Decoders

(Language models)



Encoders

(bi-directional context)



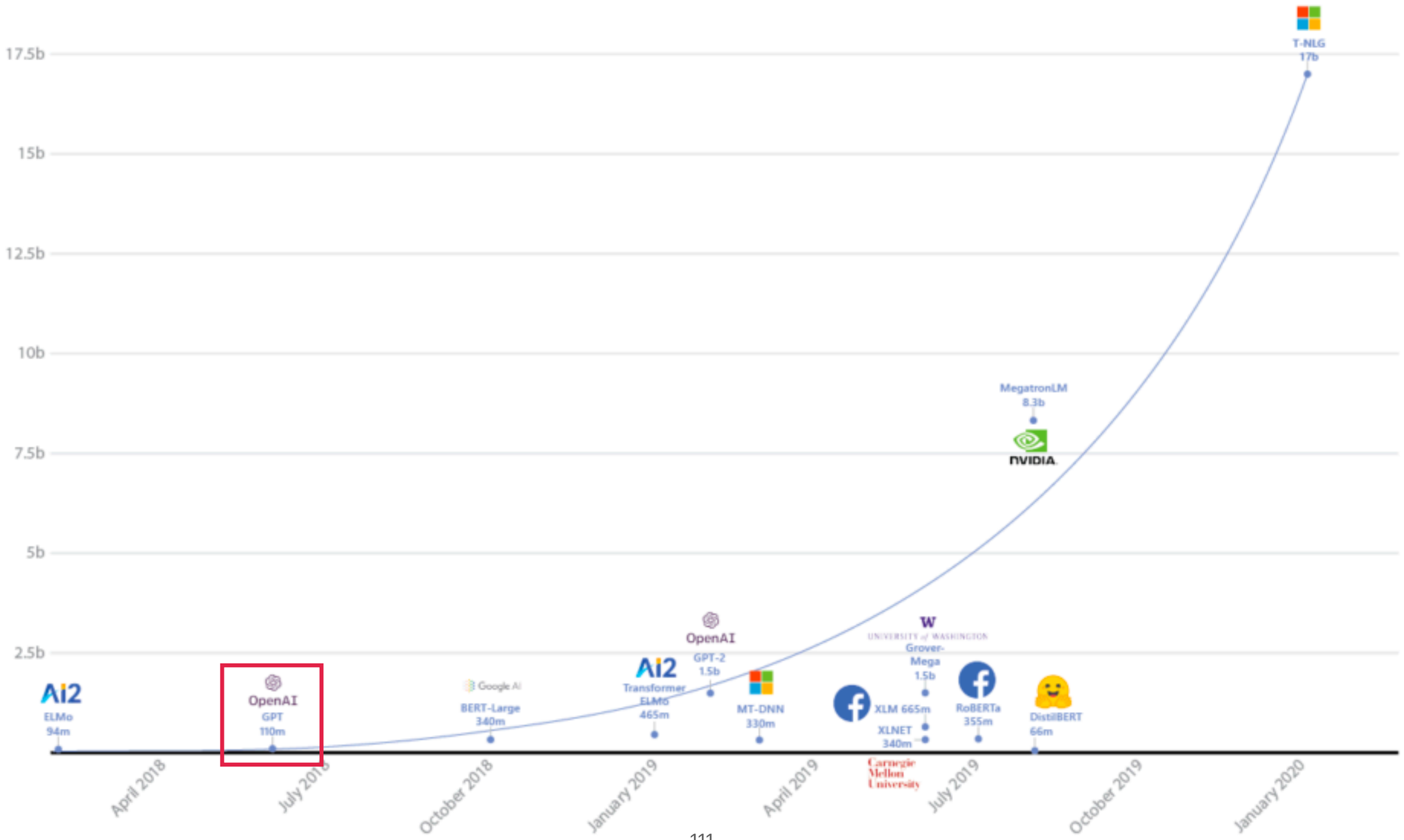
Encoders-decoders

(best of both worlds?)

Decoders

Example: GPT

- Transformer decoder with 12 layers.
- 768-D hidden states, 3072-D feed forward hidden layers.
- Trained on BooksCorpus over 7000 books.


17.5b
15b
12.5b
10b
7.5b
5b
2.5b
T-NLG
17b
MegatronLM
8.3b
NVIDIA
OpenAI
GPT-2
1.5b
Ai2
Transformer
ELMo
465m
Google AI
BERT-Large
340m
MT-DNN
330m
UNIVERSITY of WASHINGTON
Grover-
Mega
1.5b
XLM 665m
XLNET
340m
Carnegie
Mellon
University
RoBERTa
355m
DistilBERT
66m
Ai2
ELMo
94m
OpenAI
GPT
110m
April 2018
July 2018
October 2018
January 2019
April 2019
July 2019
October 2019
January 2020

Transformer decoder with 175 billion parameters

Encoders

- Trained with to guess words in masked sentences
- **Example: BERT**
  - 24 layers, 1024-D hidden states, 16 attention heads, 340 million params.
  - Trained on BooksCorpus (800 million words) and English Wikipedia (2,500 million words)
  - BERT was pre-trained with 64 TPU chips for a total of 4 days.

Encoders-decoders

- Trained to fill in sentence segments of varying size
- **Example: T5**

Summary

- Sequence-to-sequence models were a breakthrough in tasks such as machine translation
- Representing a full sentence with a single vector is a bottleneck
- Attention mechanisms allow focusing on different parts of the input and solve the bottleneck problem
- RNN-based sequence-to-sequence models require sequential computation and have difficulties with long range dependencies

Summary

- Self-attention provides an alternative to recurrent models
- Transformers are the current state-of-the-art for many tasks
- Pre-training large models and fine-tuning for downstream tasks is a very effective recipe
- Pre-training language models is a form of self-supervised learning
- Models such as ELMo, BERT, GPT, follow this procedure

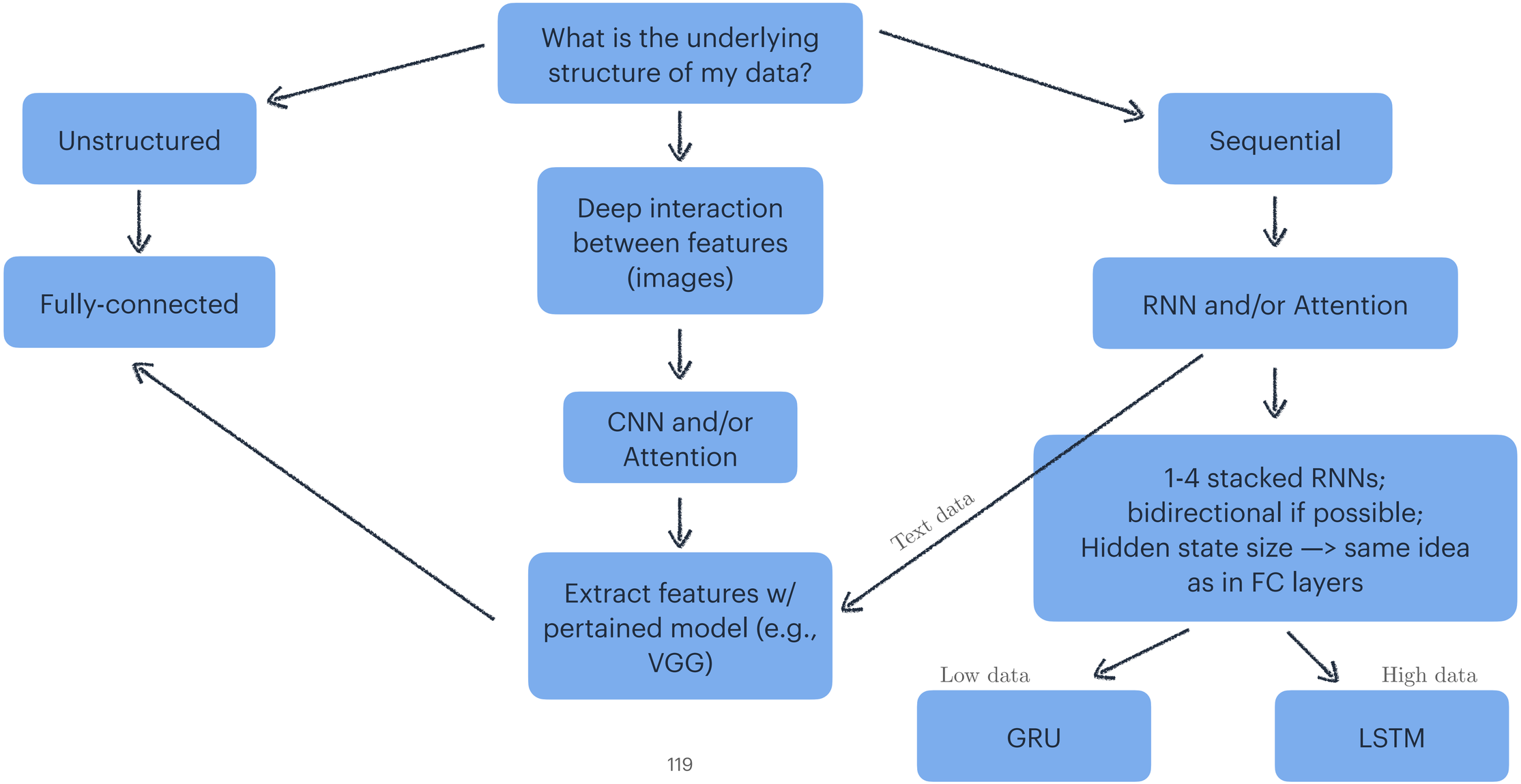
References

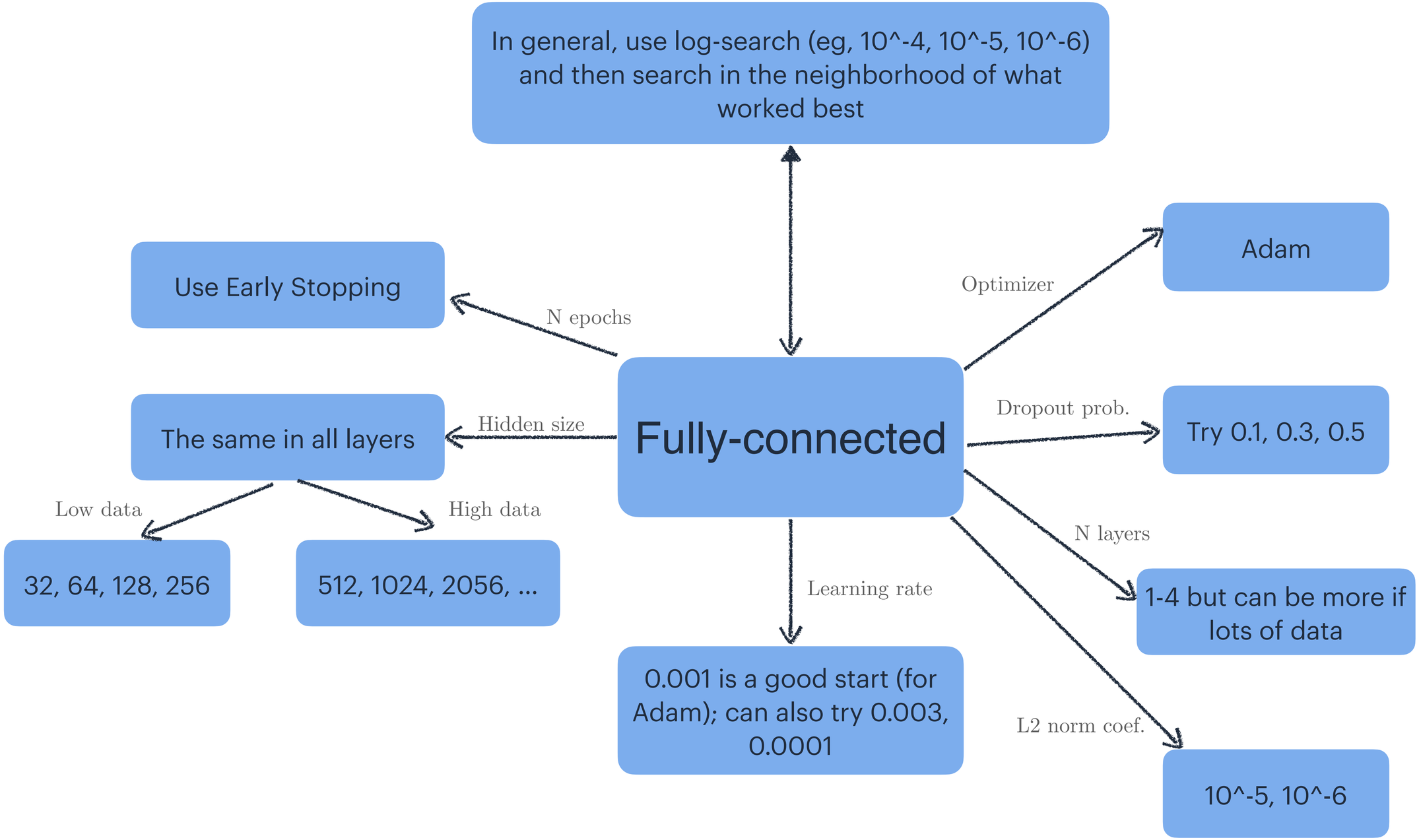
- Bahdanau, D., Cho, K., and Bengio, Y. "Neural machine translation by jointly learning to align and translate." In *Proc. Int. Conf. Learning Representations*, 2015.
- Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. "Language models are few-shot learners." *CoRR*, abs/2005.14165, 2020.
- Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. "BERT: Pre-training of deep bidirectional transformers for language understanding." *CoRR*, abs/1810.04805, 2018.
- Luong, M., Pham, H., and Manning, C. "Effective approaches to attention-based neural machine translation." *CoRR*, abs/1508.04025, 2015.
- Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. "Deep contextualized word representations." *CoRR*, abs/1802.05365, 2018.

References

- Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. "Improving language understanding by generative pre-training." Preprint, 2018.
- Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. "Exploring the limits of transfer learning with a unified text-to-text transformer." *Journal of Machine Learning Research*, 21:1-67, 2020.
- Sutskever, I., Vinyals, O., and Le, Q. "Sequence-to-sequence learning with neural networks." In *Advances in Neural Information Processing Systems*, pp. 3104-3112, 2014.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). "Attention is all you need." In *Advances in Neural Information Processing Systems*, pp. 5998-6008, 2017.

A Hitchhiker's Guide to Neural Networks

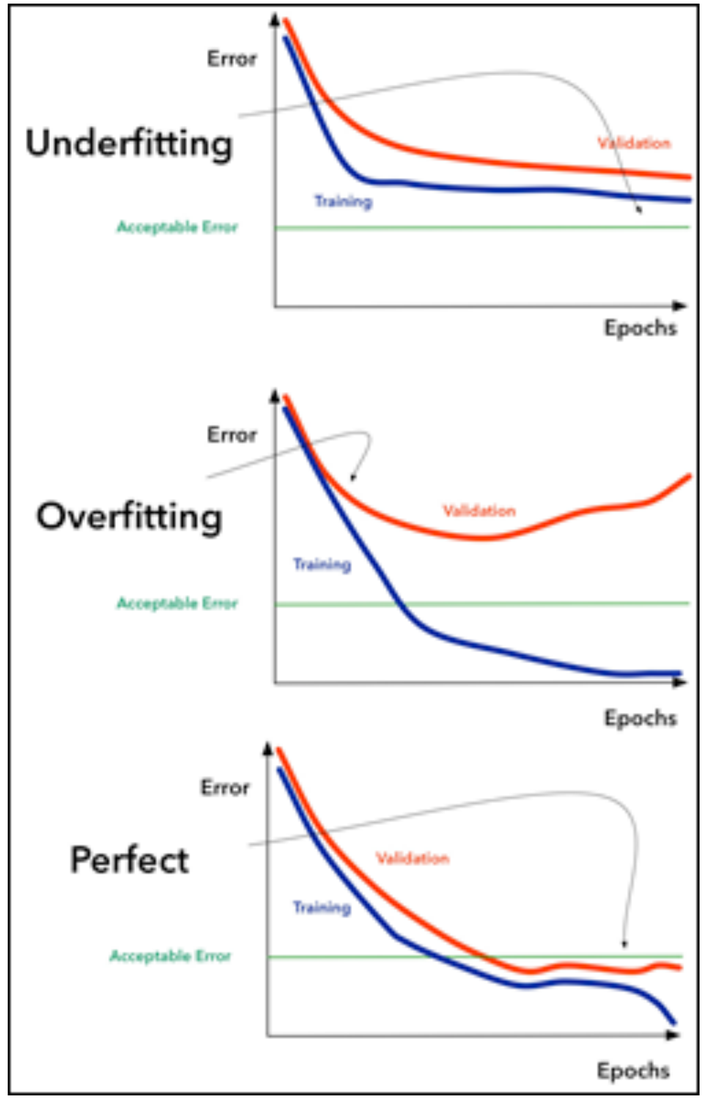



In general, use log-search (eg, 10^-4, 10^-5, 10^-6) and then search in the neighborhood of what worked best
Fully-connected
N epochs
Use Early Stopping
Optimizer
Adam
Hidden size
The same in all layers
Dropout prob.
Try 0.1, 0.3, 0.5
Low data
High data
32, 64, 128, 256
512, 1024, 2056, ...
N layers
1-4 but can be more if lots of data
Learning rate
0.001 is a good start (for Adam); can also try 0.003, 0.0001
L2 norm coef.
10^-5, 10^-6

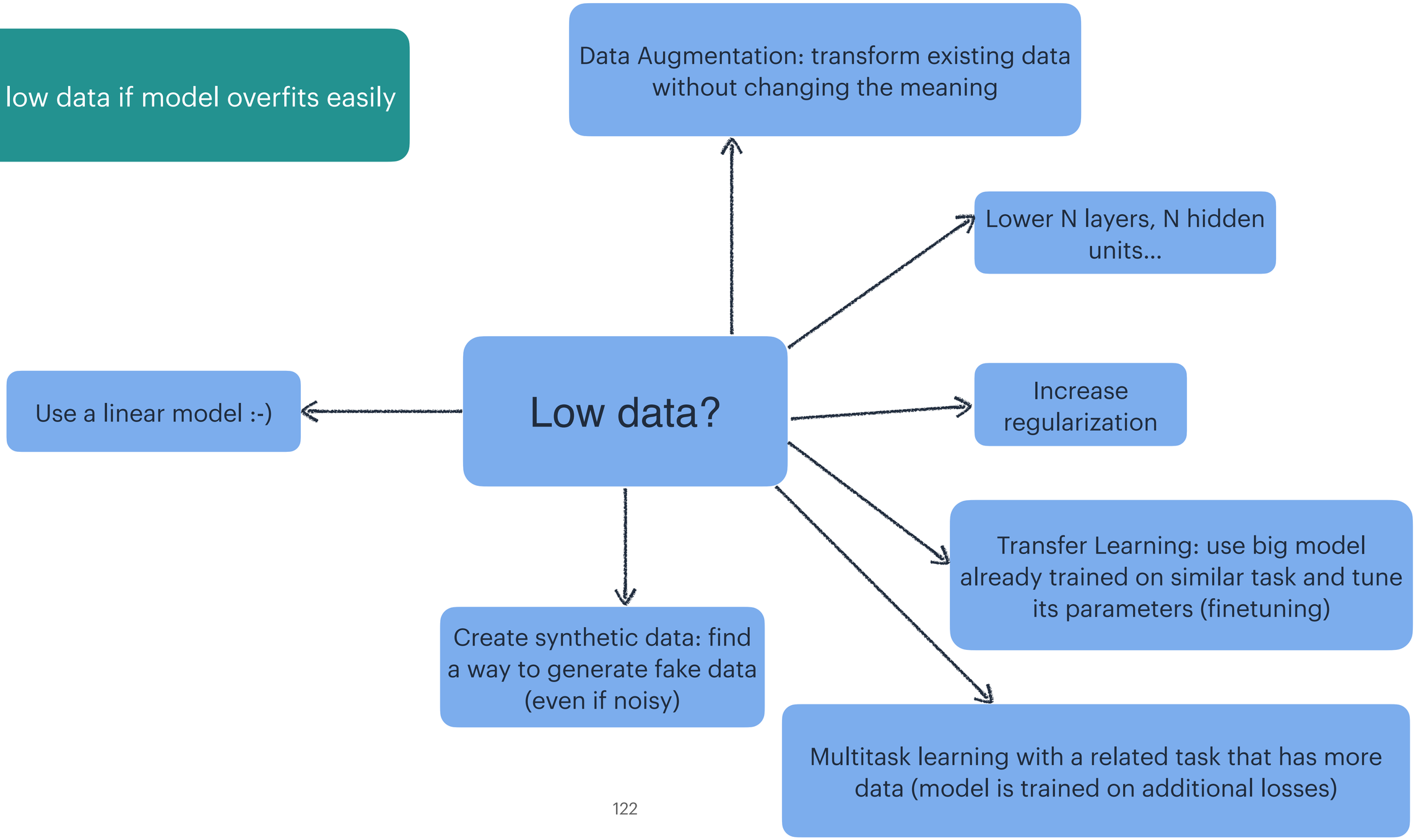
If there are no bugs in the code, network should be able to 100% overfit on a small number of training examples

Diagnostics

At each epoch, always save the training and validation performance, to later plot and make “learning curves”



There's low data if model overfits easily



More on:

- https://machinelearningmastery.com/recommendations-for-deep-learning-neural-network-practitioners/